Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

# *Run Fast!*: um jogo de corrida ubíquo dinamicamente reconfigurável

Rafael Simão da Costa

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientador
Prof.ª Dr.ª Carla Denise Castanho

Coorientador
Ms. Fabricio Nogueira Buzeto

Brasília
2013

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Coordenador: Prof.ª Dr.ª Maristela Terto de Holanda

Banca examinadora composta por:

Prof.ª Dr.ª Carla Denise Castanho (Orientador) — CIC/UnB
Prof. Dr. Ricardo Pezzuol Jacobi — CIC/UnB
Prof. Ms. Tiago Barros Pontes e Silva — DIN/UnB

**CIP — Catalogação Internacional na Publicação**

da Costa, Rafael Simão.

*Run Fast!*: um jogo de corrida ubíquo dinamicamente reconfigurável / Rafael Simão da Costa. Brasília : UnB, 2013.
109 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2013.

1. jogos eletrônicos, 2. *middleware* uOS, 3. celulares, 4. reconfigurável, 5. computação ubíqua

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

# *Run Fast!*: um jogo de corrida ubíquo dinamicamente reconfigurável

Rafael Simão da Costa

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof.ª Dr.ª Carla Denise Castanho (Orientador)
CIC/UnB

Prof. Dr. Ricardo Pezzuol Jacobi
CIC/UnB

Prof. Ms. Tiago Barros Pontes e Silva
DIN/UnB

Prof.ª Dr.ª Maristela Terto de Holanda
Coordenador do Bacharelado em Ciência da Computação

Brasília, 6 de agosto de 2013

# Dedicatória

Dedico este trabalho a meus pais, que sempre apoiam minhas decisões e me dão o suporte necessário para alcançar meus objetivos.

# Agradecimentos

Primeiramente, agradeço a minha orientadora, professora Carla Denise Castanho, que me auxiliou e guiou durante todo o desenvolvimento deste projeto. Também agradeço aos professores Ricardo Pezzuol Jacobi e Tiago Barros Pontes e Silva por terem aceito o convite para participarem da banca de avaliação deste trabalho.

Agradeço, também, ao Fabricio Nogueira Buzeto que esteve presente durante todo o desenvolvimento do projeto, auxiliando em vários aspectos, inspirando idéias para o desenvolvimento do jogo, ajudando em problemas com o *middleware*, guiando a escrita do texto, entre outros desafios.

Agradeço aos meus amigos e familiares, que deram forças e apoio nesta jornada.

# Resumo

Com a difusão de dispositivos eletrônicos e consequente aumento do número de dispositivos por pessoa, surgem possibilidades de se criar aplicações que usufruam de mais de um único aparelho. Neste contexto surge a computação ubíqua, cujos princípios estabelecem a aplicação e uso da tecnologia de modo invisível, pró-ativo e sensível ao contexto. Os jogos ubíquos se baseiam nesses mesmos princípios enquanto focam na criação de *softwares* voltados ao entretenimento em um ambiente pulverizado de dispositivos computacionais. Além disso, os jogos ubíquos se apresentam como excelentes cenários de experimentação de novas tecnologias ubíquas e pervasivas. Este trabalho consiste em desenvolver um jogo, denominado *Run Fast!*, executado em um computador com tela compartilhada e que utiliza celulares como dispositivos de entrada de dados e de execução de minijogos. Para o desenvolvimento deste jogo, foram utilizados conceitos dos jogos ubíquos, como os de computação pulverizada e sensibilidade ao contexto, de forma que, além de usufurir de vários dispositivos, o jogo se reconfigura dinâmicamente em função da entrada e saída de jogadores no jogo em tempo de execução. A implementação do *Run Fast!* utiliza o uOS, um *middleware* para ambientes ubíquos que provê mecanismos para identificação dos dispositivos móveis, bem como para comunicação entre estes e o jogo principal. Assim, o projeto tem como objetivo validar o uOS para jogos ubíquos.

**Palavras-chave:** jogos eletrônicos, *middleware* uOS, celulares, reconfigurável, computação ubíqua

# Abstract

With the diffusion of eletronic devices, raising the number of computers per person, arise possibilities to create applications that utilize more than only one gadget. In this context, arises the ubiquitous computing which principles establish the application and use of tecnology in an invisible, pro-active and context-sensitive way. The ubiquitous games base themselfs in the same principles while focus on the creation of softwares aimed to the entertainment in an enviroment pulverized with computetional devices. Futhermore, the ubiquitous games present themselfs as excellent scenarios of experimentation for new ubiquitous and pervasive tecnologies. This work consists in developing a game, denominated Run Fast!, executed in a computer with a shared screen and uses cellphones as input devices of data and to execute minigames. To develop this game, there was used ubiquitous games concepts, as pulverized computing and context-sensitive, such that, besides using many devices, the game dynamically reconfigures itself due to the entry and exit of players in the game at runtime. The Run Fast! implementation uses the uOS, a middleware for ubiquitous environments that provides mechanisms to identify mobile devices, as well as the communication between them and the main game. Therefore, the project has as objective to validate the uOS for ubiquitous games.

**Keywords:** electronics games, middleware uOS, cellphones, reconfigurable, ubiquitous computing

# Sumário

# Lista de Figuras

# Capítulo 1

# Introdução

Durante muito tempo a computação ficou restrita a pequenos grupos em universidades e grandes empresas. Na época dos mainframes, era necessário um grupo seleto e especializado de pessoas para manusear um único computador. Posteriormente, com o desenvolvimento dos computadores pessoais (PCs), a computação começou a ficar mais acessível. Atualmente, os dispositivos computacionais já se encontram pulverizados no cotidiano das pessoas, onde a proporção é de vários computadores (laptops, tablets, smartphones, desktops, etc.) para uma pessoa.

A Computação Ubíqua conhecida como *ubicomp* utiliza essa distribuição dos dispositivos móveis para tentar fazer da computação uma tecnologia transparente, ou seja, fazer com que o usuário não a perceba em sua volta como proposto por Mark Weiser [18]. Neste sentido, para fazer com que a computação seja uma ferramenta invisível no cotidiano das pessoas, é necessário que haja uma comunicação entre estes elementos pulverizados, de forma a criar ambientes inteligentes (*smart spaces*). O objetivo destes ambientes é, por meio da comunicação dos dispositivos e do desenvolvimento de aplicações, prover serviços inteligentes que auxiliem o usuário em suas atividades de forma transparente.

Ambientes inteligentes são normalmente compostos por um *middleware* e diversos outros dispositivos clientes, cada qual com suas aplicações. Ao primeiro, cabe tratar e abstrair os detalhes das camadas inferiores e orquestrar as interações entre os diversos aparelhos espalhados pelo ambiente, enquanto as aplicações devem tratar das interações realizadas junto aos usuários, de forma a utilizar da melhor maneira possível as capacidades disponíveis. Neste contexto, o *middleware* uOS, baseado na arquitetura DSOA (*Device Service Oriented Architecture*) [6], que foi utilizado no projeto viabiliza a construção de um *smart space* com foco na adaptabilidade de serviços, dando suporte ao desenvolvimento de aplicações para este ambiente dinâmico.

Em paralelo a essa crescente proliferação de tecnologia e dispositivos móveis, observa-se a rápida difusão e popularidade dos jogos eletrônicos [14]. Nas duas principais lojas online de aplicativos para dispositivos móveis, a Apple Store [12] e a Google Play [15], os jogos eletrônicos estão entre os mais comercializados.

Além do aumento do número de dispositivos móveis e da disponibilidade de tecnologia de comunicação, como 3G e *wireless*, outros recursos comuns atualmente em dispositivos móveis, como câmera, acelerômetro, sistema de localização (GPS) e tela sensível ao toque, contribuem para o desenvolvimento de jogos eletrônicos para tais plataformas [3]. Estes recursos são interessantes porque extrapolam os recursos usuais de mouse, teclado e *joys-*

*ticks*, que são geralmente utilizados em computadores e consoles e permitem experimentar novos tipos de interação.

Recentemente, como uma tendência natural, as áreas da computação ubíqua e de jogos móveis tem apresentado situações de intersecção, dando origem aos primeiros jogos ubíquos. Em *"What is going on with ubicomp games"* [4] Buzeto *et. al* apresentaram um estudo que aponta os jogos ubíquos pioneiros dentre os quais se destacam: *Uncle Roy All Around You* [8], *Pirates!* [2], *Can You See Me Now?* [1], *Hitchers* [7], *Smart Playing Cards* [16], *ARQuake* [17] dentre outros. Nessa relação de jogos e computação ubíqua o desenvolvimento de jogos ubíquos serve como um catalizador para a criação e experimentação de novas tecnologias e conceitos relacionados à computação ubíqua. Isto parte do principio de que, em geral, jogos provocam um engajamento maior dos usuários (*"players"*) e uma boa aceitação para implementação de novas técnicas e conceitos.

Este trabalho consiste em desenvolver um jogo dinamicamente reconfigurável, denominado *Run Fast!*, que é executado em um computador com tela compartilhada e utiliza celulares como dispositivos de entrada de dados e de execução de minijogos. Mais precisamente, o *Run Fast!* é um jogo de corrida cujos carros são controlados por equipes através dos celulares dos jogadores. O jogo permite até quatro equipes, sendo cada uma composta por, pelo menos, um integrante desempenhando o papel de piloto do carro. Ainda podem integrar a equipe, em tempo de execução, um copiloto e vários assistentes, cada um com seu respectivo celular. O piloto é responsável por acelerar e guiar o carro, o copiloto controla a mira do carro que pode disparar contra seus oponentes, além de liberar itens que podem atrapalhá-los. Já o(s) assistente(s) entram em ação quando recebem, em seus dispositivos, minijogos para serem solucionados durante o jogo.

Destaca-se que neste trabalho foram utilizados os conceitos de jogos úbiquos, de forma que, além de usufurir de vários dispositivos, o jogo se adapte dinamicamente frente às mudanças que ocorrem por conta da mobilidade de jogadores no ambiente onde o jogo está em execução. Assim, a implementação do *Run Fast!* utilizou o *middleware* uOS que possui mecanismos para realizar a identificação dos dispositivos que entram e saem do ambiente, além de viabilizar a comunicação entre os dispositivos e o jogo principal. Desta forma, o objetivo do jogo é validar a utilização do *middleware* uOS para o desenvolvimento de jogos ubíquos, testando e preparando suas capacidades de acordo com esses conceitos que o jogo utiliza.

O restante desta monografia está organizada da seguinte maneira. No Capítulo 2 são explicados os fundamentos teóricos relacionados ao trabalho desenvolvido, tais como, os conceitos e características da computação ubíqua e dos jogos ubíquos. No Capítulo 3 são descritos alguns trabalhos relacionados, descrevendo o funcionamento dos jogos apresentados e elucidando suas características principais. O Capítulo 4 traz a proposta do jogo *Run Fast!*, descrevendo suas características, dependências e funcionamento. No Capítulo 5 descreve-se como o jogo foi implementado explicando suas principais estruturas e relata-se brevemente os resultados de alguns testes realizados. Por fim, no Capítulo 6 conclui-se apontando as principais contribuições do jogo e os possíveis trabalhos futuros.

# Capítulo 2

# Fundamentação Teórica

Neste capítulo são apresentados os principais conceitos da Computação Ubíqua e dos Jogos Ubíquos, elucidando seus princípios e características. Quanto a Computação Ubíqua, são descritos ainda os conceitos de um ambiente inteligente e o *middleware* uOS [5] que será utilizado para organização do ambiente computacional implementado neste trabalho. Estes conceitos contextualizam e fundamentam a criação do jogo desenvolvido.

## 2.1 Computação Ubíqua

Para que uma tecnologia se torne ubíqua sua atuação não deve ser percebida pelo usuário, mas apenas a tarefa sendo realizada, seguindo a ideia de que uma boa ferramenta passa desapercebida no cotidiano das pessoas. Logo, o desenvolvimento da computação com seus elementos especializados e com alto nível de conexão entre eles fariam dela um instrumento tão ubíquo que mal seria notado, assim como é explicitado em *The Computer for 21st Century* [18] por Mark Weiser. Esta publicação comenta ainda sobre os laptops, sua portabilidade e a possibilidade de se utilizar recursos multimídia de som e vídeo na computação. No entanto, esses recursos não são suficientes para fazer a computação desaparecer.

O que a computação ubíqua busca é diferente do pretendido pela realidade virtual (RV). Enquanto a RV busca criar uma imersão do usuário num mundo digital, a *ubicomp* busca usar os elementos computacionais para manter o usuário focado em suas tarefas no mundo real. A RV faz uso de sensores e outras tecnologias para tornar todo o corpo o *input*, buscando dar ao usuário uma capacidade maior de focar em sua tarefa. No entanto, aumentar a imersão do usuário no mundo virtual não auxilia o usuário em atividades do cotidiano. A *ubicomp*, por outro lado, procura trazer a virtualização ao mundo real, fazendo com que as tarefas do dia a dia sejam assistidas por computador sem que os usuários precisem focar nelas. Um exemplo disto são os ambientes inteligentes (*Smart Spaces*) que tomam decisões de acordo com o contexto em que o usuário está inserido sem que ele necessite interagir com o sistema.

A computação ubíqua usufrui ainda da pulverização da computação, que é a difusão dos dispositivos eletrônicos nos elementos do dia a dia. Existindo vários dispositivos por pessoa, como ilustrado na Figura 2.1. Esta difusão diz respeito tanto a criação de novos dispositivos quanto a inclusão destes em objetos já existentes como camisas, tênis, óculos. Vislumbra-se, na computação ubíqua, um ambiente computacionalmente pulveri-

zado, onde os vários dispositivos distribuídos são altamente interconectados trabalhando em conjunto. Desta forma, utiliza os vários recursos existentes no ambiente para compreender e avaliar os contextos em que o usuário se encontra, podendo tomar decisões de acordo com cada um deles.

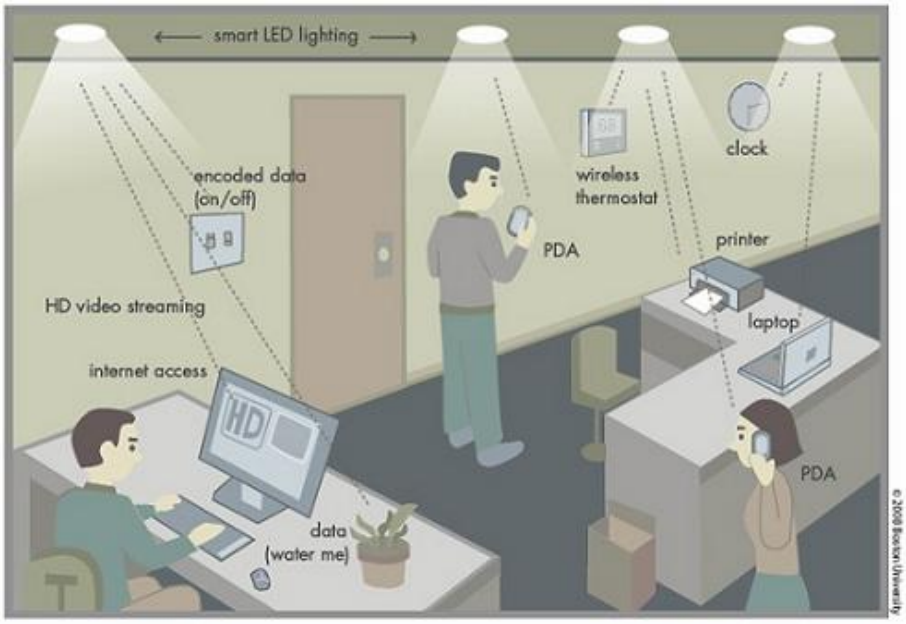



Figura 2.1: Difusão dos dispositivos eletrônicos.

Dois conceitos importantes da computação ubíqua ainda são os de invisibilidade e tecnologia calma. A invisibilidade almejada pela *ubicomp* trata de fazer com que a computação esteja muito presente no cotidiano e ainda assim ser pouco percebida, sugerindo que os computadores deveriam ser como a infância, rapidamente esquecida, mas marcando nossa vida inteira [19]. Já o conceito de tecnologia calma diz respeito ao desafio de produzir uma tecnologia que pouco atinge a atenção do usuário e que, ao atingir, seja de uma forma não intrusiva e periférica [20]. Dessa maneira, o usuário pode se concentrar melhor em outras tarefas.

### 2.1.1 *Smart Space*

Um Ambiente Inteligente ou *Smart Space* é um ambiente que possui vários dispositivos computacionais distribuídos interconectados e com aplicações inteligentes. Estas aplicações inteligentes utilizam a comunicação dos elementos presentes e seus recursos para capturar o contexto em que o usuário está inserido. Desta forma, as aplicações podem tomar decisões para auxiliar o usuário em suas tarefas.

Para construir um ambiente inteligente são necessários 3 componentes [5]:

- Hardware de baixo consumo à baixo custo;
- Uma rede que agrupe os dispositivos, possibilitando a integração dos mesmos;
- Uma aplicação (*software*) ubíqua que desfrute da integração.

#### Hardware de baixo consumo e custo

Primeiramente, existe uma necessidade de se utilizar dispositivos portáteis para facilitar o manuseio e implementar a ubiquidade. Atualmente, já existem muitos equipamentos móveis como celulares, tablets, tocadores de música. A utilização de tais dispositivos exige um hardware de baixo consumo para fornecer uma maior longevidade, uma vez que a energia desses dispositivos dependem de baterias.

Além disso, estes hardwares precisam ser de baixo custo, pois é necessário a existência de vários dispositivos por pessoa. O baixo custo permite que estes dispositivos sejam difundidos pelo ambiente. Com mais dispositivos distribuídos, o ambiente passa a ter mais capacidades computacionais e uma maior coleta de dados, facilitando a criação do ambiente inteligente.

#### Rede de comunicação

Para a formação de um ambiente ubíquo é fundamental que os dispositivos estejam conectados entre si para que possam ser organizados e coordenados. Assim suas várias capacidades computacionais podem ser agrupadas juntamente com seus dados coletados, fornecendo ao sistema ubíquo uma maior capacidade de tomar decisões. Para tal ambiente inteligente é importante a utilização de boas redes de comunicação. Isto ocorre pela necessidade de tomadas de decisões rápidas e transmissão de volume de dados entre os dispositivos.

#### Aplicações

Para que um ambiente inteligente seja criado os dispositivos têm que ser coordenados entre si por alguma aplicação. Esta aplicação deve coletar as informações dos vários dispositivos e utilizar seus recursos para fornecer serviços inteligentes aos usuários. Além disso estes serviços devem ocorrer, preferencialmente, de forma quase imperceptível ao usuário.

O fato destas aplicações executarem sobre vários dispositivos, de diferentes plataformas, de diferentes sistemas operacionais, dificulta bastante o seu desenvolvimento. Para tanto, na criação destes ambientes a construção de *middlewares* capazes de controlar as aplicações entre os dispositivos e abstrair as partes de mais baixo nível é comum. Isto ocorre por fazerem uma abstração das partes de mais baixo nível desses ambientes e facilitarem a implementação de aplicações para os mesmos.

### 2.1.2 *Middleware* uOS

O *middleware* uOS foi desenvolvido por Buzeto [5] durante seu mestrado. Este *middleware* foi construído utilizando a arquitetura DSOA (*Device Service Oriented Architecture*) [6] que é uma arquitetura que organiza um ambiente úbiquo. As aplicações que executam em um ambiente ubíquo precisam de uma organização que diga como os recursos disponíveis podem ser acessados. Isto ocorre por haverem diversos dispositivos de naturezas diferentes, que possuem recursos e serviços variados, interconectados e que podem ser utilizados por aplicações diferentes. Além disto existem outros requisitos que devem ser levados em conta no desenvolvimento de aplicações para tais ambientes. Pois

existem dispositivos com detalhes de interação diferentes, dispositivos com capacidades limitadas, com memória limitada, com baixo processamento, com bateria, entre outros.

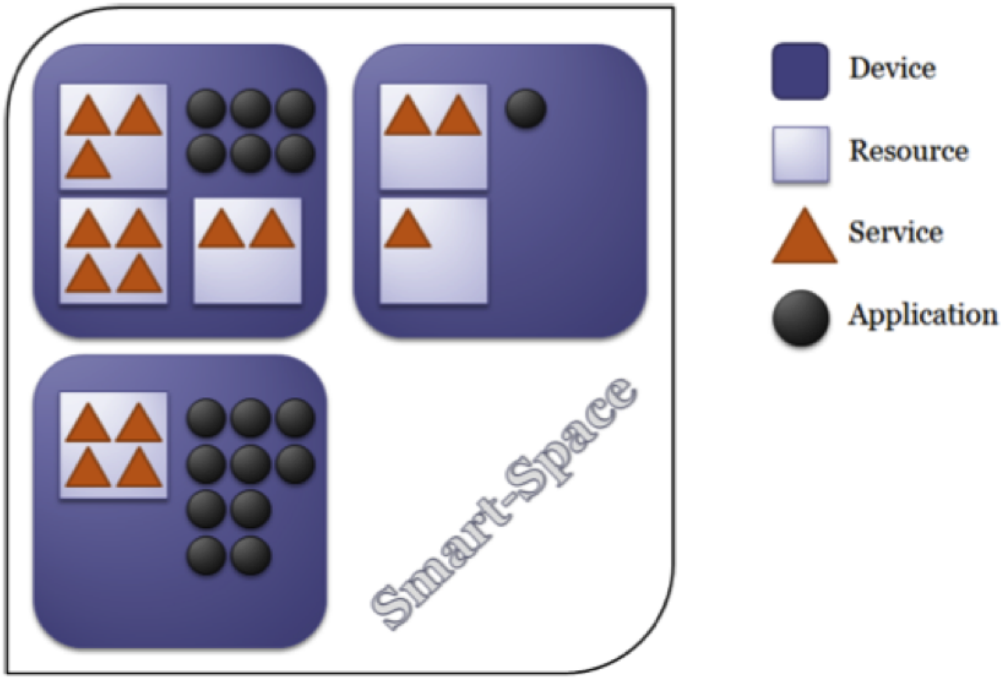



Figura 2.2: Exemplo de Smart Space da DSOA. [6]

A DSOA [6] consiste em uma extensão da arquitetura SOA (*Service Oriented Architecture*). A arquitetura DSOA utiliza de cinco conceitos, como mostrado na Figura 2.2, para organizar o ambiente e facilitar o desenvolvimento de um *smart space* e o acesso as capacidades dos dispositivos presentes:

- Ambiente inteligente: é um espaço com mais de um dispositivo com poder computacional conectados de forma colaborativa, buscando prover serviços inteligentes aos usuários.
- Dispositivo: é um equipamento com capacidades computacionais que pode se comunicar. Estes equipamentos hospedam as aplicações e compartilham suas capacidades computacionais no ambiente inteligente.
- Recurso: é um grupo de funcionalidades relacionadas que devem ser acessadas pelo ambiente através de interfaces pré-definidas.
- Serviço: é a implementação de uma funcionalidade que utiliza algum recurso e que está disponibilizada no ambiente por meio de uma interface pública conhecida.
- Aplicação: é a implementação de regras que envolvem os usuários e os recursos presentes no ambiente.

Estes conceitos estruturam o ambiente. Além deles, é necessário que haja uma capacidade do ambiente de descobrir os recursos dos dispositivos e que os estes se comuniquem

entre si, o que ocorre por meio de troca de mensagens. Para isto, é necessário que o formato dessas mensagens seja conhecido por ambas as partes, sendo necessário uma padronização das mensagens. Assim, além do uOS seguir a arquitetura DSOA ele utiliza o conjunto de protocolos uP para que haja uma interface adequada dos recursos, de forma padronizada. O protocolo uP define como os aplicativos e os recursos se comunicam. O uP fornece uma série de mensagens no formato JSON (*Java Script Object Notation*) e disponibiliza meios de troca de mensagens entre os dispositivos.

Utilizando então o uP e a arquitetura DSOA, foi desenvolvido o *middleware* uOS que fornece interfaces para reconhecer os serviços no ambiente. Este *middleware* foi desenvolvido para as plataformas JSE (*Java Standard Edition*) e Android. O uOS estabelece interação utilizando vários *plug-ins* de rede e fornece diversos mecanismos para criação de aplicações e *drivers* de serviços. Os *drivers* são os responsáveis por abstrair o funcionamento dos recursos, de forma que possam ser usados pelas aplicações e para isso possuem, então, serviços, que são as implementações das funcionalidades que o dispositivo disponibiliza no ambiente.

O uOS possui um *radar* que de tempos em tempos faz uma varredura pelo ambiente atualizando a lista de dispositivos presentes. Cada dispositivo no ambiente deve possuir uma instância do *middleware* uOS para que eles possam se comunicar. Assim, o *middleware* de cada dispositivo possui suas informações próprias, como o nome que reconhece o dispositivo no ambiente e os *drivers* e serviços que ele disponibiliza. Uma vez que dois dispositivos em um ambiente se encontrem, eles compartilham suas informações, permitindo às suas aplicações que utilizem seus serviços e troquem mensagens para se comunicar. Sendo que cada aplicação é responsável por manter a checagem da lista de dispositivos coletados pelo *radar* e tomar as devidas ações necessárias pelas mudanças destas entradas e saídas.

## 2.2 Jogos Ubíquos

Os jogos eletrônicos cresceram e se difundiram na sociedade atual. Possuem um vasto número de adeptos em praticamente todas as faixas etárias. Além disso, o crescimento desta indústria tem impulsionado o desenvolvimento de várias áreas da computação, desde processamento até mobilidade. Pode-se mencionar alguns exemplos, o desenvolvimento rápido de placas gráficas (*GPUs*), que se fez necessário para representar mundos próximos da realidade com alta definição. O surgimento dos videogames portáteis como o *GameBoy*, que estenderam as noções de onde e quando os jogos são jogados [14], levando consigo o desenvolvimento da computação móvel. O desenvolvimento da massificação dos jogos, surgindo os MMOG's (*Massive Multiplayer Online Games*) em que abrigam milhões de usuários jogando simultaneamente e em conjunto, que exigem muito das capacidades computacionais, com sistemas em nuvem e suporte a muitos usuários.

Tirando proveito deste cenário, no sentido de desenvolver e testar as tecnologias ubíquas voltadas para o entretenimento, surgem os Jogos Ubíquos (*ubigames*). Esta área é conhecida também por outros nomes como Jogos Pervasivos (*Pervasive Games*), Jogos *ubicomp* e Jogos Sensíveis ao Contexto (*Context-Aware Games*), embora haja quem os diferencie em determinados aspectos [11].

Os Jogos Ubíquos buscam trazer o jogo para a vida real, de forma transparente, fazendo que uma cidade inteira seja o mapa do jogo, por exemplo. Tem como objetivos também a

questão de usufruir de dispositivos computacionais disseminados no ambiente para criar os serviços inteligentes necessários. Observando os *ubigames* já desenvolvidos, podemos perceber características semelhantes entre eles quanto ao Ambiente, a Flexibilidade de Jogadores, ao Contexto e a Interação com o Usuário [4].

### 2.2.1 Ambiente

O ambiente se refere ao tipo de local onde as sessões de jogo ocorrem. Estes podem acontecer *Indoor* ou *Outdoor*, ou seja, em ambientes Fechados (*Indoor*) que são ambientes limitados, dentro de salas por exemplo, ou em ambientes Abertos (*Outdoor*) em que se extravasa o jogo para ambientes mais amplos, como cidades, onde as limitações são inexistentes.

Os ambientes *Indoor* permitem um controle maior das ações do jogador (*player*), tendo mais informações sobre as interações do usuário. Neste tipo de ambientes é comum a utilização de sensores, até mesmo para usufruir do fato do ambiente ser fechado para ser mais preciso quanto as atitudes do player. No entanto, geralmente, há uma limitação quanto ao número de participantes no jogo.

Em ambientes *Outdoor*, embora o jogo tenha menos controle sobre as interações do player, o jogo acaba tendo uma imersão maior no mundo real, justamente pelo fato de estar em um ambiente mais amplo. Isto está relacionado também com a utilização da localização do jogador via GPS ou até mesmo pela informação dada pelo usuário para tomar decisões no jogo. Além disso, estes jogos costumam ser mais livres quanto ao número de jogadores.

### 2.2.2 Flexibilidade de Jogadores

Em geral, espera-se uma maior flexibilidade da quantidade de jogadores num Jogo Ubíquo, gerando até mesmo a expectativa de alcançar um número de participantes próximo aos números dos MMOG's, principalmente para os jogos *Outdoor*. No entanto, esta expectativa nem sempre é alcançada e estes jogos podem ainda ser divididos em 4 tipos principais semelhantes aos jogos comuns: podem ser Individuais, de Time, de Múltiplos Jogadores ou Colaborativos [4].

Tanto em jogos individuais como em jogos de múltiplos jogadores, os usuários tem objetivos pessoais e suas ações em geral afetam apenas a si próprios, sendo diferenciado apenas o número de jogadores que podem jogar ao mesmo tempo. Nos jogos ubíquos individuais, geralmente, busca-se ter um controle maior das ações do usuário. Por estes jogos ter apenas um jogador por vez, torna-se mais fácil fazer análises sobre o jogo. Já os jogos de Múltiplos Jogadores, que é o mais comum em jogos abertos, possibilita uma flexibilidade do número de jogadores, embora o controle sobre cada jogador seja menor.

Quanto aos jogos de times e os colaborativos, ambos possuem uma boa flexibilidade no que diz respeito a quantidade de jogadores. No entanto, os de times dividem os *players* em grupos com objetivos em comum e possuem algumas regras quanto a interação dos participantes. Já nos colaborativos cada jogador tem um objetivo e eles podem se ajudar durante o jogo, sendo mais flexíveis quanto as regras de interação e ocorrem geralmente em ambientes abertos.

### 2.2.3 Contexto

O contexto é obtido observando as ações dos usuários, que pode ser por meio de sensores e câmeras, por exemplo. Estes dados de contexto podem estar relacionados à identidade do usuário de modo que dependendo de quem está jogando o jogo funcione de forma diferente. Outra alternativa é que o jogo tome decisões ou se modifique em função da localização do *player*, que pode ser obtida via GPS, triangulação ou até mesmo pelas informações dadas pelo jogador. Também é possível buscar e levar em consideração dados relacionados ao estado do usuário, tanto física quanto emocionalmente. Isso pode ser feito usufruindo de medições de batimentos cardíacos, temperatura, e qualquer outra informação biológica que possa ajudar no desenvolvimento do jogo.

### 2.2.4 Interação com o Usuário

A interação com o usuário diz respeito a forma como o jogador se comunica com o jogo. Esta interação pode ser dada tanto de formas mais tradicionais, usando mouse, tela e teclado, como de formas mais elaboradas, com sensores e câmeras. Além disso, essas interações podem ser diretas ou indiretas. Interações diretas estão mais relacionadas a *inputs*. Já as indiretas estão mais ligadas a dados obtidos indiretamente, como dados de localização obtidos via GPS, por exemplo.

# Capítulo 3

# Projetos Relacionados

Neste capítulo são apresentados alguns trabalhos realizados na área de jogos ubíquos, deixando mais claro os conceitos mostrados ao longo do texto e empregados no desenvolvimento do projeto. São mostrados dois jogos em ambientes abertos, um em semiaberto e um em ambiente fechado, ilustrando os diferentes tipos de jogos úbiquos.

## 3.1 *Uncle Roy All Around You*

*Uncle Roy All Around You* [8] é um jogo que mistura elementos de teatro em um jogo de investigação, que aconteceu no final de Maio e início de Junho de 2003 no centro da cidade de Londres. Seu cenário inclui uma praça e um parque da cidade, se caracterizando como um ambiente aberto. Isso, somado ao fato de se integrar jogadores na rua com jogadores online permite uma flexibilidade colaborativa entre eles.

Jogadores online inicialmente tem acesso a um site com informações do jogo e como jogar. Além disso, pode-se entrar no jogo, no entanto existe um limite de 10 jogadores por vez. Ao entrar, eles se encontram numa representação 3D do espaço do jogo, onde eles podem ver as mensagens trocadas no chat, cartões dos jogadores reais com informações deles, além da posição destes. Os jogadores online tem informações que os jogadores da rua não possuem como a posição do escritório do *Uncle Roy* e podem se comunicar com os jogadores na rua, podendo ajudar ou atrapalhar eles.

O jogo começa quando os *players* que ficam na rua entram no jogo com um *ticket* de uma hora. Depois disso, é pedido que eles deixem os pertences no local e recebem em troca um PDA (*Personal Digital Assistants*). O PDA é usado para se comunicar com o *Uncle Roy* e receber instruções de como proceder durante o jogo. Depois disso um ator dá a primeira instrução a esta pessoa informando-a que a missão dela é encontrar o *Uncle Roy* e ensina-a a utilizar a interface do jogo no PDA. Feito isto a primeira tarefa do player é de procurar pela cidade o ponto vermelho num mapa no PDA. Quando o jogador chegar ao local ele deve declarar sua posição ao *Uncle Roy* marcando o local de sua posição no mapa e utilizando o botão *"I am here."* no PDA. Sempre que o usuário declara sua posição o *Uncle Roy* manda uma mensagem pré-programada de volta com novas intruções. A maioria das informações dadas são úteis, no entanto algumas são apenas pistas enganosas, como pedir para o jogador que siga alguém. Além disso os jogadores na rua, podem receber mensagens de jogadores online que podem estar acompanhando seu progresso e responder com pequenas gravações de áudio.

No fim do jogo, alguns jogadores chegam a uma porta e recebem o comando de tocar a campainha, neste momento então o PDA que eles receberam e estão carregando dá algumas instruções pré-programadas que levam os jogadores a um apartamento vazio onde lhes é pedido para escrever um *post-it* respondendo a pergunta *"When can you begin to trust a stranger?"*, que quer dizer "Quando você pode começar a confiar num estranho?". Em seguida, eles são instruídos à esperar do lado de fora na cabine de telefone, onde o telefone toca e um ator que está ligando pede para que a pessoa espere do lado de fora do prédio. Finalmente, depois disso, aparece uma limusine perguntando se desejam entrar e caso aceitem o convite os leva de volta ao início enquanto são feitas algumas perguntas sobre acreditar em estranhos, uma vez que durante o jogo se recebe várias informações de pessoas desconhecidas, e se eles se candidatariam para ajudar outros jogadores.

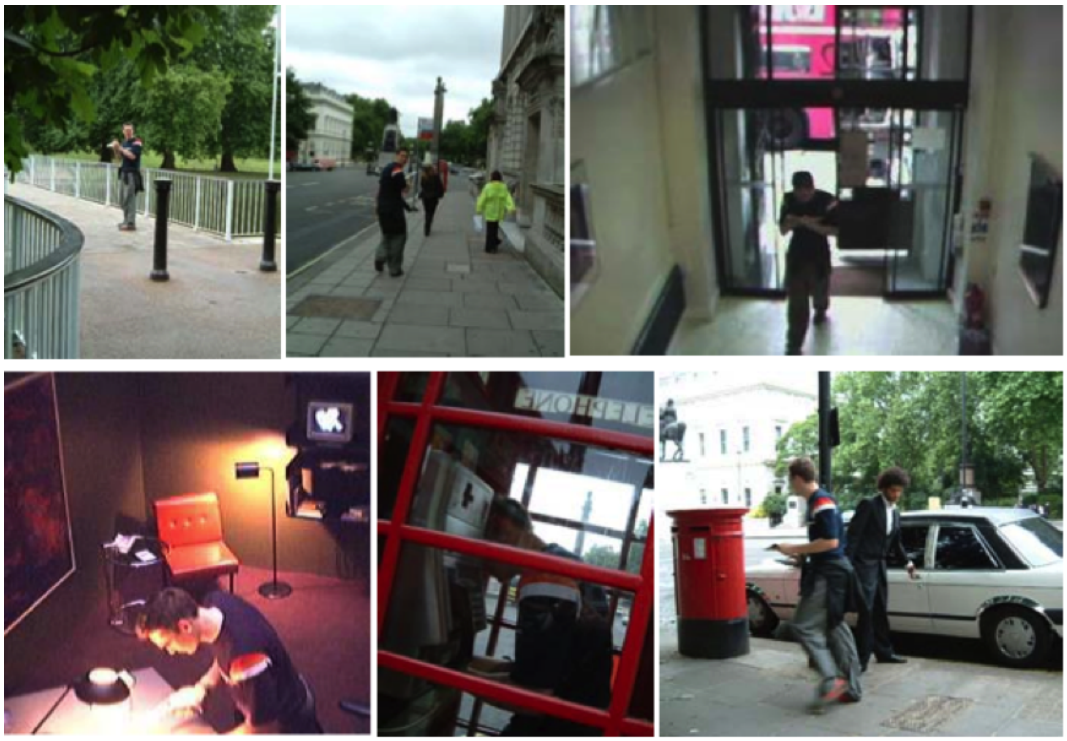


Figura 3.1: Um jogador na rua experienciando o *Uncle Roy All Around You*. No parque, nas ruas da cidade, entrando no escritório, dentro do escritório, na cabine de telefone e na limusine [8].

Este jogo mostra várias formas para combinar teatro e jogos, encorajando os jogadores a ultrapassarem os limites do comportamento seguro na cidade. Os jogadores encaram uma tensão por se sentirem sozinhos numa tarefa sem muitos guias, no entanto a apreensão acaba gerando uma sensação de se virar sozinho raramente gerada nas pessoas. Além disso, gera-se uma experiência constrangedora pelo fato de se misturar aspectos da vida real com outros elementos pré-programadas. Essa mistura gera experiências poderosas, mas difícil de serem estabelecidas para grandes públicos, uma vez que é necessário um pessoal dedicado ao jogo e um certo controle dos jogadores. O jogo também foi usado para validar um método de coleta de localização auto-informada, sendo que neste caso os usuários foram bem honestos.

Encontrar formas de aumentar o suporte à uma quantidade maior de participantes neste jogo, mostra-se um importante ponto de pesquisas futuras. No *Run Fast!*, procura-se resolver o problema de dar suporte a um grande número de jogadores através da divisão de personagens. Sendo que uma vez que todos os jogadores principais (pilotos e copilotos) estiverem sido selecionados, os novos participantes podem se juntar aos times como assistentes de forma ilimitada.

## 3.2 *Hitchers*

*Hitchers* [7] é um jogo para celular que almeja criar experiências com sensibilidade ao contexto baseado em localização, sendo um exemplo de jogo *Outdoor* com múltiplos jogadores. Este é um jogo de celular cujo objetivo é levar caroneiros virtuais até seu destino. O jogo utiliza a informação do posicionamento do celular para interagir com os caroneiros. O jogo foi construído como um aplicativo padrão de cliente-servidor, o cliente executa no celular do usuário do Nokia Series e um servidor comum com extensões de *PHP* e *MySQL*. O sistema de localização utilizou rede GPRS usando as cell-Ids dos jogadores para localizá-los e a partir deles verificar os *hitchers* que podem ser pegos e carregados, fazendo-os andar de célula em célula. Os jogadores podem criar, pegar ou deixar caroneiros. Os caroneiros digitais podem ser criados pelos jogadores dando a eles nomes, registrando uma pergunta, um objetivo e um rótulo para informar a posição atual (Figura 3.2).


Figura 3.2: Criando caroneiros (*hitchers*) [7].

Os jogadores que forem pegar um andarilho procuram pelos que estão por perto e o jogo procura na célula em que eles estiverem presentes, que é ilustrado na Figura 3.3. Depois pegam um caroneiro, respondem a pergunta, os levam até uma nova posição e os deixam colocando um rótulo para informar onde o caroneiro está. Permitindo que o jogo aprenda a posição de tal estabelecimento. Caso o jogador não encontre nenhum mochileiro na célula em que ele se encontra o jogo pode dar-lhe dicas de locais que possuem caroneiros. Por fim, os jogadores podem acessar o website do jogo e visualizar as informações dos andarilhos que eles deram carona e os que eles criaram. Além disso, pode-se ver os locais que o andarilho esteve, quem deu carona e quais foram as respostas para a pergunta,

como representado na Figura 3.4. Assim os jogadores podem acompanhar o destino dos andarilhos que criaram.


Figura 3.3: Procurando caroneiros [7].

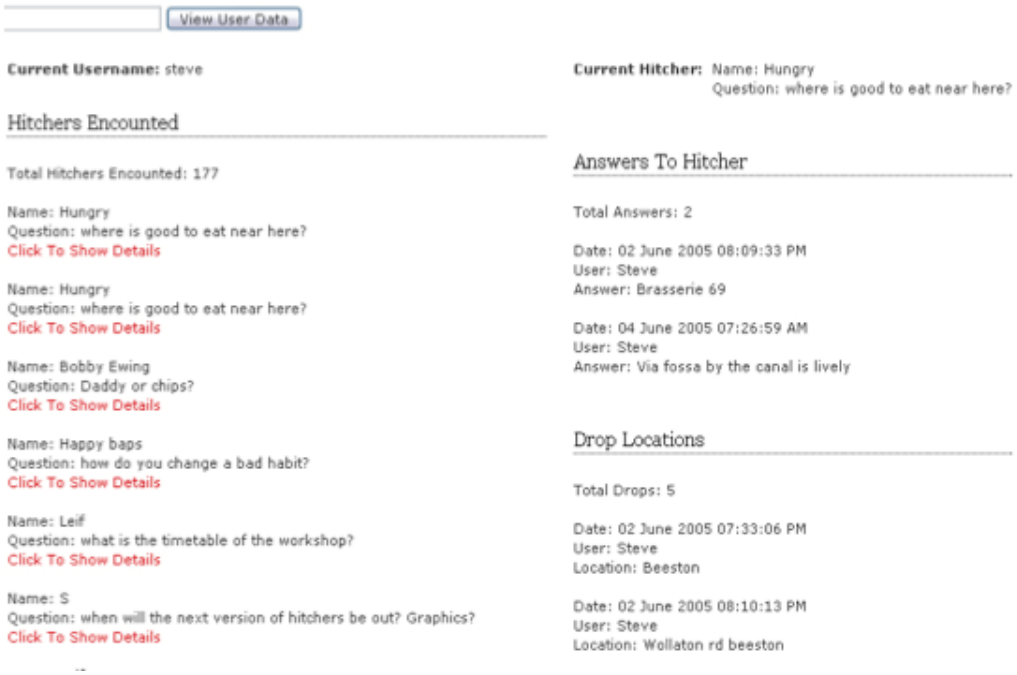



Figura 3.4: Explorando o histórico de caroneiros [7].

Este jogo traz a utilização de dispositivos móveis para indentificar cada jogador e usufrui das informações neste dispositivo, no caso as informações de localização, para modificar os acontecimentos dentro jogo. Seguindo esta mesma estratégia, o *Run Fast!* utiliza as informações de cada celular para identificar os jogadores e diferenciar as ações de cada um, mais precisamente, diferenciando as ações dos jogadores de acordo com a informação de qual personagem este representa.

## 3.3 *ARQuake*

O objetivo deste jogo é investigar como converter um jogo de primeira pessoa de desktop para um jogo *outdoor/indoor* utilizando realidade aumentada. ARQuake [17] é um jogo de FPS (*First Person Shooter*), que utiliza a localização do jogador, como referência dentro do mapa do jogo, e realidade aumentada, buscando provocar uma grande imersão no jogo. O jogo utiliza um equipamento especialmente montado para obter as interações do usuário. Foi criado um mundo correspondente a um campus de uma universidade possuindo assim características tanto de um campo aberto, quanto as limitações de locais fechados sendo considerado um jogo *Indoor/Outdoor*. Este usa como base o jogo *Quake* pelas seguintes razões:

- Possui gráficos compatíveis com uma grande variedade de sistemas operacionais e dispositivos;
- É de código aberto, permitindo utilizá-lo na contrução de novos jogos;
- Possuir todos os elementos necessários para um FPS 3D já desenvolvidos.

O jogo base possui dois objetivos básicos: sobreviver e fugir do local em que se está. O jogo apresenta várias criaturas de diferentes formas, velocidades e ataques. Estes monstros procuram atacar o jogador, tirando sua vida (*life*), e o player, no entanto, possui algumas armas que utiliza para matar esses monstros.

O objetivo do *ARQuake* era desenvolver a abordagem de Realidade Aumentada (RA) para um jogo *Indoor/Outdoor*. Para isso foi montado um equipamento específico para se jogar este jogo. Foi utilizada uma mochila onde foram colocados um notebook com o sistema operacional *LinuxOS*, um compasso digital utilizado para a orientação, uma arma com dois botões para realizar as ações de atirar e mudar de arma, uma *Head Mounted Display* (HMD) para visualizar os elementos do jogo e GPS para saber o posicionamento do jogador. Este equipamento está sendo apresentado na Figura 3.5.

Figura 3.5: Equipamento montado específicamente para jogar ARQuake. [17].

Para fazer o rastreamento da posição do jogador na fase foram utilizadas, além do GPS e do compasso digital, também marcadores fiduciais. Estes marcadores foram distribuídos em locais estratégicos pelo campus como referência. Desta forma, o GPS é utilizado para um rastreamento de maior granularidade do posicionamento, e assim no momento em que se reconhece a presença de um marcador utiliza-o como referência de posicionamento. Quanto aos monstros que apareceriam no jogo, foi feita uma seleção das criaturas que não fossem muito rápidas e que atacassem de longe, excluindo também os que fossem muito grandes e os que tivessem efeitos inesperados. Esta seleção foi feita principalmente por causa das limitações dos dispositivos de rastreamento do posicionamento do jogador. Além disso, havia uma grande quantidade de processamento por causa da Realidade Aumentada, que acaba deixando o jogo lento.

Foi criada uma fase de *Quake* representando o campus *Mawson Lakes* da Universidade da Austrália do Sul, ou seja, uma modelagem da representação do campus. Nesta fase as paredes do jogo são exatamente as paredes internas do campus e o delimitador externo da universidade. As paredes modeladas não são renderizadas pela máquina gráfica do jogo o que faz com que as paredes sejam as paredes vistas na vida real. Sendo assim, apenas os monstros, itens e os dados do jogador são renderizados na tela. Foram distribuídos pela fase 200 monstros e vários itens com armaduras, itens de cura, armas. De forma a deixar o jogo interessante e não forçar a máquina gráfica.

Com o sistema executando, o jogador se movimenta pela fase andando e modifica a visualização movimentando o olhar (Figura 3.6). Na parte inferior da visão se encontra as informações de status e as informações da arma que está sendo utilizada. Muitos usuários reportaram o jogo como sendo intuitivo e fácil de jogar, no entanto, certas vezes foi difícil mirar no alvo por causa da diferença de tempo entre a ação e a apresentação dos elementos do jogo.

Figura 3.6: Display acoplado à cabeça de um jogador [17].

Embora em menores proporções, a criação de um equipamento próprio para jogar este jogo influenciou na ideia de utilizar o celular do jogador para identificá-lo no jogo *Run Fast!* além de adaptá-lo para ser também o dispositivo de entrada do jogo.

## 3.4 *Smart Playing Cards*

O motivador deste jogo [16] é o desafio da computação ubíqua de criar ambientes inteligentes que sejam úteis e, ao mesmo tempo, mantenham as regras sociais. Para tal, desenvolveu-se uma versão aumentada do jogo de cartas *Whist* já existente, utilizando dispositivos computacionais. Este foi escolhido para ser o jogo a ser desenvolvido pelo fato do sistema usado para detectar as cartas nas mesas possuir um limite no número de cartas simultâneas. Neste jogo, utiliza-se no máximo 4 cartas por rodada. Além disso, *Whist* não necessita de comandos pronunciados como UNO e possui a possibilidade de se implementar algumas funcionalidades para incrementar o jogo, tais como, mostrar o vencedor, a pontuação de cada time, dicas para iniciantes.

O *Whist* é um jogo dividido em dois times, onde cada time possui 2 jogadores. Utiliza um baralho comum com 52 cartas, sendo que inicialmente distribui-se 13 cartas para cada um dos jogadores, a última carta entregue define a cor da carta de trunfo. O jogo é dividido em rodadas onde o jogador a direita do participante que entregou as cartas começa a primeira rodada. Cada jogador desce uma carta por rodada e ganha a rodada o jogador que descer o maior trunfo ou a maior carta no caso de não haver nenhum trunfo. O jogador que vencer a rodada começa a próxima, por fim o time que tiver ganho o maior número de jogadas ganha o jogo.

O jogo utiliza um sistema de identificação por frequência de rádio (*Radio Frequency Identification* - RFID). Cada uma das 52 cartas possui uma etiqueta com um identificador próprio. A mesa de jogo possui um leitor RFID ligado a um computador. Como interface com os usuários temos uma tela de acesso público e PDAs pessoais. A Figura 3.7 representa a organização dos dispositivos.

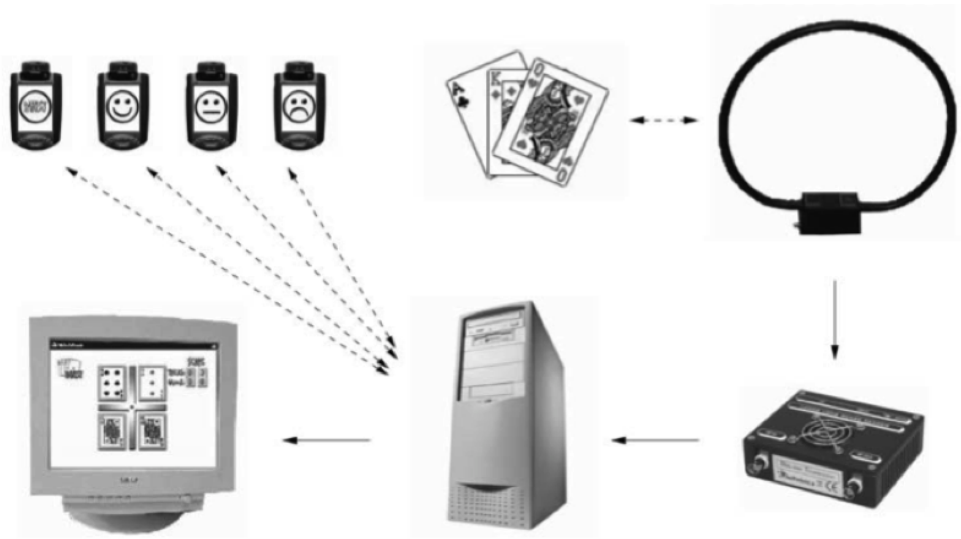


Figura 3.7: Organização dos dispositivos no *Smart Playing Card* [16].

Na tela do computador aparecem informações referente a todos os jogadores como o placar e o vencedor. Nos PDA's que podem ser utilizados um para cada jogador aparecem dados referentes ao jogador em questão apenas como uma avaliação do nível da jogada feita por ele. Os PDA's se comunicam com a aplicação que está executando no computador, via *wireless*.

A aplicação controla a vez de cada um, sendo assim se alguém descer antes de sua vez é disparado um alarme avisando o erro e perguntando se deve-se corrigir o erro. Aparece na tela as cartas jogadas na rodada e o placar com o número de rodadas ganhas por cada time. Nos PDA's um sistema verifica se a jogada foi boa ou não baseado nas cartas que o jogo sabe que já foram jogadas, uma vez que o sistema guarda todas as cartas que foram jogadas no jogo, inclusive de outras rodadas. O jogo verifica de acordo com as cartas na mão do jogador se a jogada foi boa ou não, dando um *feedback* de uma carinha feliz, triste ou indiferente. A Figura 3.8 apresenta uma representação de como os elementos interagem.

Figura 3.8: Representação de como os elementos interagem no *Smart Playing Card* [16].

Este é o jogo com que o projeto do *Run Fast!* mais se identifica, compartilhando a preocupação de criar um jogo num ambiente inteligente mantendo as relações socias. Além de utilizar uma organização dos dispositivos semelhantes, onde cada jogador possui um dispositivo móvel que se comunica com um computador que une e agrupa as informações do jogo.

# Capítulo 4

# O Jogo *Run Fast!*

O jogo *Run Fast!* foi projetado para ser executado dentro de um contexto de computação ubíqua, onde usuários entram e saem de um ambiente portando seus dispositivos móveis. Baseado nisso, o jogo procura usufruir dos dispositivos distribuídos pelo ambiente para criar uma experiência inovadora. Além disso, como foi observado no Capítulo 2, os jogos ubíquos propiciam um contexto experimental para novas tecnologias relacionadas à computação ubíqua.

O jogo desenvolvido neste trabalho é um jogo de corrida divido em equipes, que é executado em um computador cujo *display* é compartilhado por todos os jogadores e cuja entrada de dados é realizada através do celulares do jogadores.

O jogo foi construído de modo a favorecer a colaboração entre os jogadores e explora os dispositivos presentes no ambiente para permitir isso. Além disso, a divisão dos jogadores em times provoca, naturalmente, a interação desejada. Isto colabora com o intuito de desenvolver um jogo ubíquo que satisfaça o desafio de manter as relações sociais.

Outra característica relevante do *Run Fast!* é a capacidade de se adaptar dinâmicamente aos dispositivos e usuários presentes no ambiente, ou seja, jogadores podem entrar e sair do ambiente e, consequentemente, do jogo, durante sua execução. O próprio jogo identifica e convida um novo usuário através do seu dispositivo para participar do jogo. Além disso, o jogo percebe quando um jogador deixa o ambiente e reconfigura o jogo para o contexto atual de jogadores presentes.

A seguir, é apresentado o *game design* do jogo *Run Fast!*, abordando as questões referentes ao conceito, escopo e mecânica do jogo.

## 4.1 Conceito

*Run Fast!* é um jogo de corrida no qual seu aspecto visual lembra as corridas de um Autorama, porém com vários incrementos. Este jogo foi projetado para executar num ambiente ubíquo, utilizando celulares com sistema operacional Android para operar as ações como se fossem controles de *input* e realizar alguns minijogos. Além disso, a parte principal do jogo deve executar em um computador com uma tela que possa ser visualmente compartilhada pelos jogadores.

Os jogadores podem ser distribuídos em no máximo 4 times onde cada time possui no mínimo 1 jogador. Os participantes do grupo podem ser um dos três personagens do jogo: o piloto, o copiloto ou o assitente. Cada um dos personagens possui tarefas e habilidades

diferentes e realizam ações diferentes no jogo. O piloto é responsável por dirigir o carro coletando os itens e esquivando o carro dos ataques e armadilhas do inimigo. O copiloto possui a capacidade de atirar projéteis e deixar bombas pela pista, causando danos ao inimigo e atrapalhando seu percurso. O assistente ajuda o time com os possíveis minijogos que podem ocorrer durante a corrida, tendo apenas que dar suporte ao time. É permitido um piloto, um copiloto e vários assistentes, sendo que a presença do piloto é obrigatória na equipe. Para o jogador interagir com o jogo é necessário o uso de um celular. Sendo este um jogo cooperativo e dinâmico a comunicação verbal entre os membros de uma equipe é incentivada para se alcançar maior eficiência durante a sessão.

O objetivo de cada time é ganhar a corrida e para isso deve-se coletar o máximo de dinheiro possível e dar o maior número de voltas para conseguir somar mais pontos que o adversário. Além disso, existem os objetivos específicos de cada personagem. O piloto tem como objetivo dar o maior número de voltas conduzindo o carro e coletando itens para lhe ajudar. O copiloto tem como objetivo atrapalhar os times inimigos soltando bombas e atirando para causar danos aos inimigos. Os assistentes tem como objetivo coletar o maior número de moedas no minijogo Bônus e terminar o mais rápido possível o minijogo *Break* para que o time possa continuar a corrida.

A corrida ocorre em uma pista padrão (*default*) retangular com 2 a 4 carros. Quando o tempo de jogo acabar, vence o time com a maior pontuação, que é igual ao número de voltas multiplicado por 100 somado a quantidade de dinheiro que o time obteve. A performance do carro é determinada pelo trabalho em equipe realizado pelo grupo. O cenário do jogo está representado na Figura 4.1.

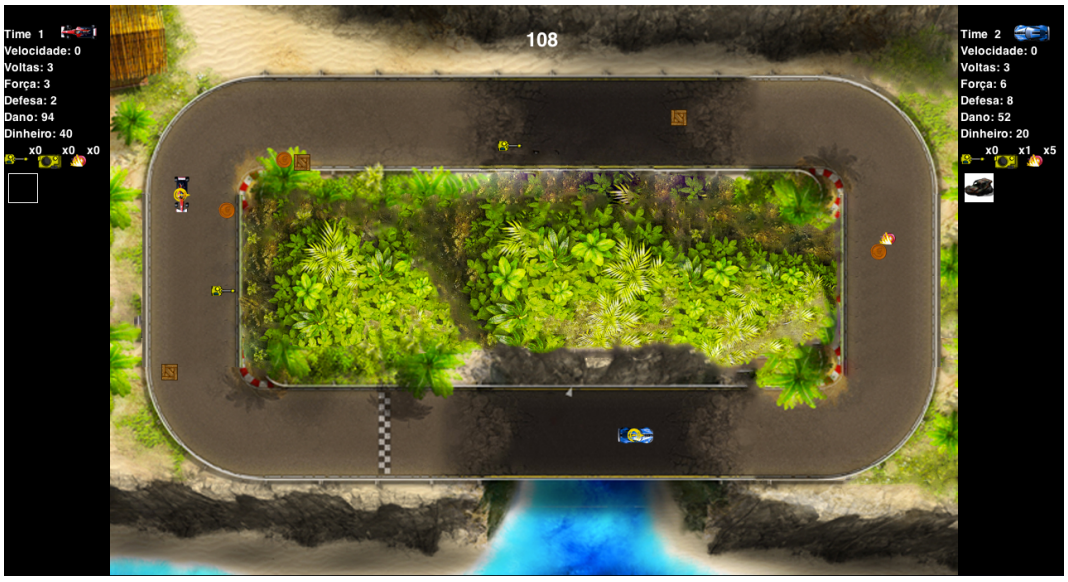



Figura 4.1: Representação do cenário do jogo para uma corrida de 2 equipes (carros).

Os minijogos estão relacionados aos itens coletados durante a corrida que são ativados pelo piloto. Alguns itens correspondem a um minijogo, e caso o piloto ative-o, os jogadores que o receberem deverão resolvê-lo para voltar ao jogo principal. Caso apenas o piloto esteja jogando em uma equipe, e ele receba um minijogo, o carro dele deverá parar até que ele finalize o minijogo e assim possa retornar a corrida. Existem dois minijogos: o Bônus e o *Break*. O Bônus é um jogo simples em que se aparecem moedas aleatoriamente na tela

do celular durante um certo tempo e o número de moedas ganhas aumentam o dinheiro do grupo. Já o *Break* é atribuído a um dos times adversários que devem concertar o carro para poderem voltar a corrida.

Este é um jogo dinâmico que procura promover uma interação mais próxima entre os jogadores do que em jogos de videogames comuns, utilizando dispositivos de diferentes naturezas. Além disso, busca-se estabelecer uma capacidade de reconfiguração baseada no número de jogadores, onde os jogadores podem entrar enquanto a corrida ocorre e o número de jogadores interfere nas possíveis ações do time e na facilidade de concluir os minijogos durante a disputa.

## 4.2 Detalhamento

A mecânica do jogo define as principais características do jogo como os componentes essenciais, quais os conflitos e suas soluções, além do fluxo do jogo. Além disso é importante dar uma visão geral de como estes elementos se integram no jogo.

### 4.2.1 Visão Geral

Os jogadores se conectam ao jogo via celular, uma vez conectado o jogador escolhe um dos atores do grupo de forma que o piloto deve ser sempre escolhido e apenas um jogador pode escolher o copiloto, todos os que entrarem a partir daí serão assistentes. Caso um jogador chegue no meio da partida, o jogo percebe sua chegada através do *middleware* uOS e o convida para jogar. Ao aceitar o convite o usuário pode entrar em um time já existente, uma vez que não há limite para participantes de uma equipe, ou criar um novo time, respeitando o número máximo de quatro equipes. Caso o jogador resolva criar uma nova equipe ele receberá uma tela de seleção onde deverá escolher um carro, em seguida será atribuído a ele o piloto e o carro aparecerá na linha de partida na corrida que já deverá estar acontecendo.

A velocidade dos carros é determinada pelo acelerador que o piloto possui. Um contador de voltas de cada time é incrementado cada vez que o carro ultrapassa a linha de partida. O carro tem uma quantidade de pontos de vida, que diminuem com colisões e ao receber tiros ou passar em cima de bombas ou armadilhas. Quando os pontos de vida acabam o carro explode e perde todos os equipamentos. O time deve esperar 5 segundos e então o carro volta a corrida. Todos os atributos referentes aos carros podem ser modificados com os equipamentos coletados durante a corrida.

Os itens e os equipamentos do jogo aparecem aleatoriamente pela pista de corrida e para coletá-los basta passar por cima deles. Se coletar um equipamento, este modifica os atributos do carro, e se coletar outro referente ao mesmo atributo, seu efeito é aplicado novamente, não havendo restrição. Caso colete um item, ele vai para a caixa de item do time, no entanto um time pode ter apenas um item por vez e se a caixa já estiver ocupada não se pode pegar outro até que o atual seja usado.

Uma vez que os jogadores estejam conectados e o jogo inicie, tem-se a interface da tela geral compartilhada por todos os jogadores e as interfaces personalizadas para cada personagem (piloto, copiloto e assistente) nos celulares. Além disto, nas laterais da tela principal utilizada por ambos os times encontram-se as informações gerais de cada um dos times, como velocidade, número de voltas dadas, a conservação do carro, o poder de

força do carro, a defesa, o dinheiro, os itens e os equipamentos. No jogo de corrida, o jogo principal, o cenário é uma pista retangular com um ponto de partida na parte inferior da pista. Os personagens do jogo são os carros cada um referente a um time que começam no ponto de partida e completam uma volta cada vez que passam novamente por este ponto.

Para cada jogador, há uma interface personalizada no celular que varia de acordo com o personagem que ele está vinculado. O piloto possui o acelerador que aumenta a velocidade do carro, os botões que permitem ele dirigir o carro, além da possibilidade de liberar armadilhas coletadas na pista. O copiloto possui a possibilidade de atirar usando uma mira que fica em cima do carro e gira em torno do seu eixo e pode deixar bombas na pista. Os assistentes auxiliam nos minijogos que forem disparados durante o jogo e são executados nos celulares.

A fase bônus é, na verdade, um item bônus e deve ser ativado para ser jogado. O piloto deve ativá-la para que o minijogo seja repassado aos integrantes do grupo que deverão resolvê-lo para poder retornar ao jogo principal. Esta fase bônus é um jogo de coleta de moedas, em que aparecem moedas aleatoriamente na tela do celular durante um tempo pré-determinado e quanto mais moedas forem coletadas mais dinheiro o time ganha. Assim que os integrantes terminam a fase bônus estes retornam ao jogo principal, no caso, à corrida.

O outro minijogo existente é o jogo denominado *Break*. Ele é representado por um item que ao ser acionado pelo piloto dispara um minijogo a um dos times adversários com o intuito de atrapalhá-los. Todos os integrantes do time que receber este minijogo devem resolvê-lo o mais rápido possível para poderem voltar a corrida. Enquanto isso, o carro conduzido pelo piloto deste time, tem o limite máximo da sua velocidade reduzido até que os demais integrantes finalizem o minijogo. Caso o piloto seja o único integrante do time, o carro irá parar e o piloto deve resolver o mini-jogo sozinho. Este mini-jogo consiste de um pequeno jogo onde se ganha pontos por clicar nos parafusos fora de lugar que vão aparecendo até conseguir uma quantidade mínima para voltar à corrida.

### Diferenciais

Comparando com outros jogos de corrida, este se diferencia por ser um jogo de equipes e por permitir causar dano ao inimigo. A distribuição dos jogadores em equipes traz uma abordagem diferente, uma vez ques estes jogos geralmente são jogos individuais. Além disso, a capacidade de atrapalhar o adversário, causando-lhe dano, é uma característica incomum para os jogos do gênero.

A reconfiguração do jogo de acordo com a quantidade de jogadores é um diferencial quando comparado com outros jogos úbiquos. A capacidade de saber quem está presente no ambiente e perceber quem entra e sai permite que o jogo utilize esta informação para se modificar. Neste caso de acordo com a quantidade de jogadores, o número de itens e equipamentos gerados e o tempo se modificam de modo a suportar uma corrida com mais competidores. Além destes diferenciais, o jogo utiliza celulares como *input* das ações dos jogadores e apresenta alguns minijogos que são diparados para serem resolvidos durante a corrida.

### 4.2.2 Componentes Essenciais do Jogo

#### Atributos dos Carros

Os carros operam como avatares dos jogadores na pista. É através deles que pode-se observar a reação dos comandos durante uma sessão. Essa reação é determinada pelo conjunto de atributos que se segue:

- **Velocidade:** A velocidade de um carro é alterada pela interação do jogador com o comando do acelerador. Cada tipo de carro possui um valor máximo de velocidade que pode ser alcançada. Apesar disto, este pode ser modificado ao longo do jogo adquirindo equipamentos de velocidade.
- **Número de Voltas:** Um contador armazena o número de voltas do carro. O contador começa em zero e é incrementado cada vez que o carro ultrapassa a linha de partida no sentido anti-horário e decrementado caso passe no sentido horário.
- **Poder de Ataque:** O poder de ataque é o dano causado ao bater em um inimigo. Numa batida ambos os carros recebem dano. O dano final é a subtração do poder de ataque com a defesa do inimigo. O dano final é subtraído dos pontos de vida do inimigo, sendo 1 (um) o menor valor de dano.
- **Defesa:** A defesa do carro diminui o dano recebido pelas batidas que ocorrerem durante o jogo. A defesa diminui o dano causado inclusive pelas batidas nas laterais da pista.
- **Vida (Conservação do Carro):** A vida do carro é o estado de conservação deste. Todos os carros começam com um valor de vida igual a 100. Os pontos de vida são subtraídos pelos danos finais causados por batidas, tiros, bombas e armadilhas. Uma vez que a vida chegue a 0 (zero) o carro explode, perde todos os equipamentos e fica 5 segundos fora da corrida. Ao término dos 5 segundos ele reaparece onde explodiu.
- **Dinheiro:** A equipe ganha dinheiro nos minijogos bônus e coletando as moedas que aparecem no meio da pista. O dinheiro é utilizado para obter pontos extras ao fim da corrida.
- **Equipamentos:** Em geral um time pode ter quantos equipamentos conseguir. Cada equipamento aumenta um determinado atributo do carro. Os equipamentos são demonstrados por uma figura do equipamento e ao lado um número que define quantos o time possui.
- **Itens:** Os itens aparecem aleatoriamente na pista e podem ser adquiridos passando por cima deles. O carro tem apenas um espaço, conhecido como *slot*, para itens. Sendo assim pode-se coletar apenas um item por vez e e é possível obter outro somente quando utilizar o item coletado.

#### Equipamentos

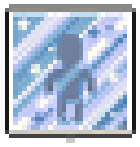
Os equipamentos incrementam os atributos do carro, sendo que cada um modifica um atributo diferente. Existem basicamente três equipamentos, a saber:

| | |
|---|---|
|  | **Reforçador:** Aumenta a defesa do carro, reduzindo a efetividade dos danos recebidos. |
|  | **Turbo:** Aumenta a velocidade máxima do carro. |
|  | **Potencializador:** Aumenta o poder de ataque do carro, causando mais dano em batidas aos inimigos. |

Os equipamentos aparecem aleatoriamente no meio da pista de corrida e podem ser adquiridos passando por cima deles assim como os itens. Uma vez que um equipamento foi obtido é incrementado o contador referente a ele no inventário do time. Não existe uma limitação para a quantidade de equipamentos sob a posse de uma equipe. Ao se capturar um equipamento, seu efeito é aplicado sobre o carro daquela equipe, no caso de se obter um equipamento repetido, o efeito é multiplicado. Por exemplo, caso um carro já possua um 'Reforçador', que aumenta em 10 sua defesa, caso adquira um segundo terá sua defesa incrementada em 20.

### Itens

Os itens ao contrário dos equipamentos são elementos acionáveis. Estas ações podem atrapalhar o time adversário, mantendo-os ocupados ou diminuindo suas capacidades, ou pode ainda liberar os minijogos. Os itens podem ser os seguintes:

| | |
|---|---|
|  | **Desequipar:** Remove os equipamentos de posse da equipe adversária anulando os efeitos fornecidos por estes; |
|  | **Armadilha:** Deixa uma armadilha na pista na qual quem passar por cima leva dano e fica temporariamente mais lento; |
|  | ***Break*:** Ocupa a equipe adversária com um minijogo, sendo que os participantes só podem voltar ao jogo quando o completarem; |
|  | **Bônus:** É um minijogo que a equipe que o possui pode jogar para poder ganhar dinheiro extra. |

Os itens aparecem aleatoriamente no meio da pista e podem ser coletados passando-se por cima deles, assim como os equipamentos, no entanto, são menos frequentes. Cada equipe tem um espaço (*slot*) disponível para itens, ao passar por cima de um item este

ocupa o espaço disponível. Ao usar um item este libera o espaço que estava ocupando. Uma vez que o espaço esteja ocupado não se pode coletar mais nenhum item até ser utilizado.

### Personagens

Os jogadores de cada equipe devem escolher um dos personagens que compõe o grupo para jogar, sendo que existem três tipos de personagens no jogo: o piloto, o copiloto e o assistente. Cada jogador de uma equipe deve selecionar um dos personagens, sendo que o piloto é o único que deve estar sempre presente, além disso, apenas o assistente pode ser selecionado por mais de um participante de sua equipe.

Embora o objetivo de todos dentro de uma equipe seja o mesmo, que é conseguir a maior pontuação, cada um dos personagens é responsável por algum determinado aspecto e, portanto, executa ações diferentes dos outros dentro do jogo. Logo, as interações dos jogadores com o jogo são distintas porque é fornecida uma interface específica para cada personagem. A ideia do jogo é permitir que os jogadores colaborem entre si, de forma que as capacidades distintas se complementem para conseguir atingir o objetivo final. A seguir, são explicadas as funções de cada um separadamente.

### Piloto

O piloto é o único personagem que deve aparecer sempre em todos os times. Ele é responsável por, através do celular, pilotar o carro, podendo bater no carro adversário, desviar das armadilhas, desviar dos tiros do adversário, coletar itens e coletar equipamentos, além de ser quem aciona os itens coletados, seguindo os comandos do controle representado na Figura 4.2.

As interações que o piloto pode realizar são as seguintes:

- **Acelerar o carro:** Através de um botão que funciona como um acelerador que mantém o aumento da velocidade até uma velocidade máxima, baseado no tempo em que o botão é pressionado (botão azul);
- **Desselecionar:** Botão utilizado para desselecionar o carro escolhido na tela de seleção caso mude de idéia de qual carro deseja jogar (botão verde);
- **Movimentar o carro:** Através de botões de movimentação que rotacionam o carro direcionando seu caminho (setas para cima, baixo, esquerda e direita);
- **Acionar os itens:** Os itens coletados estão em espaços reservados para seu armazenamento, sendo que para liberar algum basta que clique sobre o botão de seleção de item (botão caixa). Caso seja necessário escolher um time para receber os efeitos do item, ao acionar o botão de seleção de item aparecerá no display do jogo principal (em baixo da caixa de item do time) o número do time que deverá receber os efeitos do item. Para mudar entre as opções deve-se utlizar os botões de navegação dos itens (setas ao lado do botão caixa), uma vez escolhido deve-se clicar mais uma vez sobre o botão de seleção de item (botão caixa) para ativar os efeitos sobre aquele time.

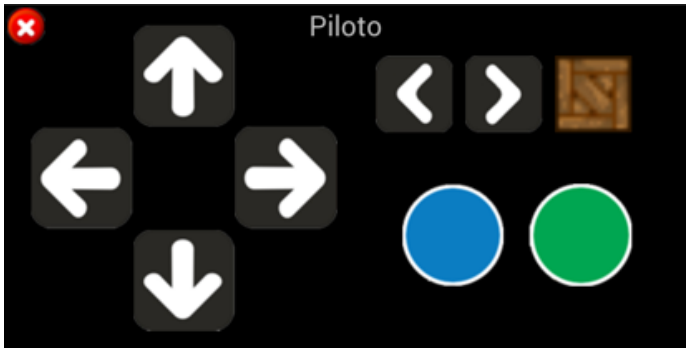



Figura 4.2: Representação do controle do piloto na tela do celular.

### Copiloto

O copiloto é o personagem responsável por atrapalhar o inimigo e dificultar seu trajeto. O copiloto pode atirar e deixar bombas pelo caminho, seguindo os comandos do controle representado na Figura 4.3. As interações que esse personagem pode fazer então são as seguintes:

- **Rotacionar a mira:** Através de botões que giram a mira que fica sobre o carro (setas para cima, baixo, esquerda e direita) de forma semelhante a como o piloto rotaciona o carro.
- **Atirar:** A mira indica a direção do tiro, e ao pressionar o botão de atirar (botão azul) saem tiros da posição atual do carro na direção referente a mira. Os tiros seguem por essa direção até acertarem um carro ou saírem do mapa, caso o tiro acerte o carro, este recebe o dano referente ao poder de ataque do atirador. Não existe limite para o número de tiros, apenas um pequeno tempo de espera entre os tiros;
- **Deixar bomba:** Um botão que possibilita deixar bombas na posição do carro (botão verde). Há um tempo de espera após se soltar uma bomba, para poder soltar outra. Quando um carro passa por cima de uma bomba este perde um determinada quantidade de vida.

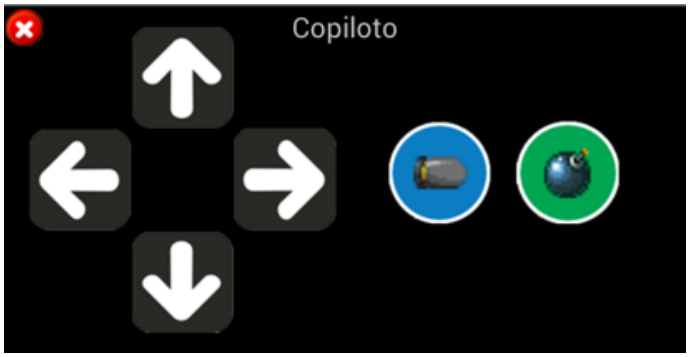



Figura 4.3: Representação do controle do copiloto na tela do celular.

### Assistentes

O número de assistentes por equipe é ilimitado, já que estes tem sua participação limitada à resolução dos minijogos que ocorrerem durante a corrida. Quanto mais assistentes um time possuir, maior são as suas chances de êxito nestas situações.

### Minijogos

Os minijogos correspondem a eventos disparados por itens durante a corrida. Estes possuem mecânicas próprias que interferem com o andamento da sessão. Existem dois tipos de minijogos disponíveis: O Bônus e o *Break*. O Bônus visa favorecer o time que ativou o item relacionado, enquanto o *Break* busca comprometer o time adversário.

### Bônus

O objetivo deste minijogo é conseguir a maior quantidade de pontos possível coletando os itens que aparecem aleatoriamente pela tela. O minijogo tem um tempo pré-determinado, que vai diminuindo até zero quando este termina. Neste tempo, aparecem aleatoriamente ao longo da tela várias moedas caindo e se misturado a essas moedas aparecem também algumas bombas. Para coletar esses elementos o jogador deve clicar sobre eles. O jogo tem um contador que acumula as moedas coletadas. No entanto, o jogador perde pontos caso pressione uma bomba. Este jogo está representado na Figura 4.4.

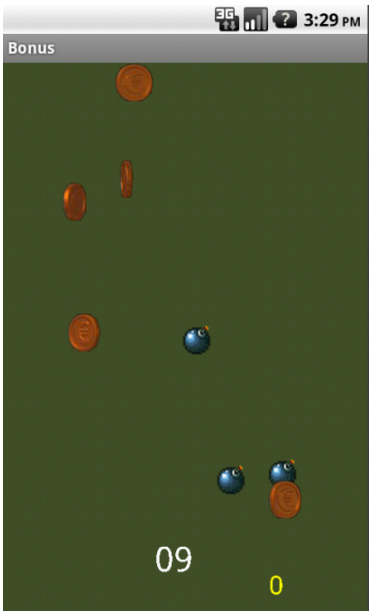



Figura 4.4: Representação do minijogo Bônus que é executado no celular do(s) assistente(s).

Este minijogo é ativado pelo piloto do time ao acionar o item Bônus. Assim que acionado este minijogo é disparado para os celulares de todos os jogadores do time, exceto o piloto. Caso o piloto seja o único jogador do time, ele recebe o minijogo e o carro fica parado até o piloto terminá-lo. Os jogadores que recebem o minijogo em seu celular tem

suas ações interrompidas imediatamente e recebem a tarefa Bônus. Assim que o jogador concordar o jogo começa, não havendo necessidade de iniciarem todos juntos.

Os elementos básicos deste jogo são:

- **Moedas:** Aparecem aleatoriamente caindo pela tela numa velocidade aleatória. São os itens mais comuns no jogo e dão uma certa quantidade de pontos ao serem coletadas, para coletá-las basta clicar sobre elas enquanto estiverem visíveis;
- **Bombas:** Assim como as moedas, aparecem aleatoriamente caindo pela tela numa velocidade aleatoria. Este elemento é menos comum que a moeda e caso seja coletado ele faz com que o jogador perca pontos. Para coletá-lo basta clicar sobre ele enquanto estiver visível.

O jogo termina quando o tempo do minijogo acaba, então os pontos coletados são adicionados a quantidade de dinheiro da equipe, que no fim da partida serão contados como pontos adicionais aos pontos conseguidos pelo número de voltas.

### *Break*

Para a corrida o objetivo deste minijogo é atrapalhar o time adversário. Já o objetivo do minijogo é coletar as partes do carro acumulando pontos, enchendo barra de progresso até concluí-la. O número de pontos diminui de acordo com o número de jogadores do time. Logo, quanto mais jogadores a equipe tiver mais fácil será concluir o minijogo. Caso a equipe tenha mais de um jogador, o piloto entra no modo piloto automático enquanto os demais jogadores resolvem o minijogo "*Break*"nos seus celulares. Caso a equipe possua apenas um jogador, no caso o piloto, o carro irá parar até que o piloto conclua o minijogo no celular. O modo piloto automático, basicamente reduz o limite máximo de velocidade do carro da equipe. Este minijogo está representado na Figura 4.5.

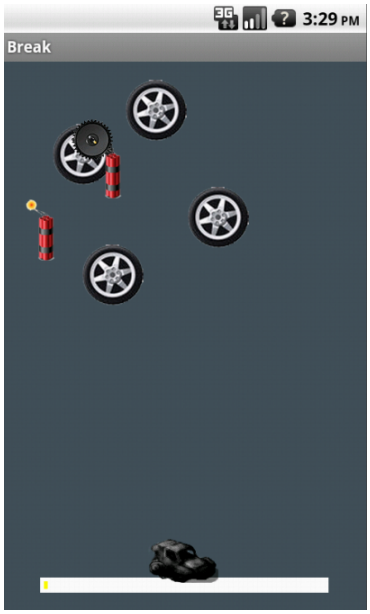



Figura 4.5: Representação do minijogo *Break* executado no celular.

Ao acionar o item *Break* um time adversário é selecionado e este minijogo é atribuído a eles para resolvê-lo. Quando este item é ativado a equipe adversária recebe o minijogo em seu celular e interrompe imediatamente o que estava sendo executado e atribui a eles a tarefa *Break*, sendo que só poderão voltar ao jogo principal quando terminá-lo.

### 4.2.3 Conflitos e Soluções

Os desafios apresentados aos jogadores durante o jogo são dos mais variados tipos, sendo que cada jogador pode enfrentar ainda conflitos específicos que outros podem não ter que lidar. Inicialmente os principais desafios enfrentados pelos jogadores são aqueles referentes a um jogo de corrida, que é o fato de ter que ser mais rápido que o adversário, além de ter de enfrentar coletivamente os vários problemas oferecidos pelos inimigos, que são os jogadores dos outros times. Alguns destes conflitos que podemos citar são: desviar, proteger e resistir aos tiros, bombas e armadilhas e dificultar os trajetos do inimigo.

Os jogadores podem ainda enfrentar desafios referentes aos minijogos, que podem ser a capacidade de resolver pequenos problemas coletivos da maneira mais rápida e eficiente possível, além da capacidade de coletar o maior número de itens, conseguindo mais recursos para o time. Os jogadores também devem avaliar os recursos disponíveis para comprar os itens corretos e melhorar suas capacidades.

### 4.2.4 Fluxo de Jogo

O jogo possui quatro estados diferentes: o menu principal, o menu de seleção dos carros, a corrida em sim e o menu de fim de partida. O menu principal é o estado inicial do jogo, onde são aprensentadas duas opções a de jogar e a de sair do jogo. O menu de seleção mostra os carros do jogo e dá aos pilotos a possibilidade de escolher o carro que utilizaram na corrida. Uma vez escolhidos os carros, inicia-se o estado da corrida em si, que é a única fase do jogo que consiste apenas do circuito retangular com cantos arredondados. Por fim, o último estado é o de fim de partida onde são apresentados o ranking dos jogadores e as opções de reiniciar a corrida, ir ao menu principal e sair do jogo.

Somado aos estados do jogo, apresenta-se, ainda, as duas principais características dinâmicas. Primeiramente, jogadores podem entrar e sair do jogo em qualquer um dos estados, permitindo uma adaptação mais fluída da jogabilidade. Assim, o jogo se adapta de acordo com a entrada e saída dos jogadores, principalmente durante a corrida, onde a entrada de um novo time gera um acréscimo de 10 segundos ao tempo do jogo, além da quantidade de itens e equipamentos gerados pela pista ser proprocional a quantidade de times presentes. Além disso, existem também os minijogos que são disparados nos celulares dos jogadores, estabelecendo rupturas temporárias do fluxo básico. Estes minijogos são ativados por itens, sendo que o minijogo *Break* se reconfigura de acordo com a quantidade de integrantes em um time, sendo que quanto mais membros o time possuir, menor a quantidade de peças de carro devem ser coletadas para terminar o minijogo.

### 4.2.5 Controles

Todas as entradas do jogo são realizadas através dos celulares dos jogadores. Além disto, eles são utilizados para interagir com os minijogos que ocorrem durante a corrida.

# Capítulo 5

# Implementação do *Run Fast!*

Os jogos eletrônicos geralmente são desenvolvidos de maneira a segregar duas camadas: a camada da *engine* e a camada de código específico. Isto para permitir uma maior reutilização do código e criar uma melhor divisão entre os componentes do jogo. As *engines* são mais independentes e podem servir de base para a construção de diversos jogos. Elas agrupam funcionalidades de abstração de componentes de hardware, de componentes lógicos, de manipulações gráficas, mixing de áudio. Já o código específico é desenvolvido sobre a *engine*, visando implementar as regras específicas do jogo. A arquitetura do *Run Fast!* segue esta distribuição em camadas. Neste trabalho, foi desenvolvida uma *engine* própria, que além de abstrair uma parte da representação gráfica e controlar as etapas do jogo, ainda dá suporte aos *inputs* dos dispositivos móveis (celulares).

O jogo *Run Fast!* é composto por 4 aplicações diferentes que interagem entre si:

1. **Jogo principal**: que é o *Run Fast!* em si, é onde a corrida ocorre e onde se concentram os dados e comandos vindos dos celulares conectados ao jogo. Este faz o papel de *Host* do jogo como um todo.
2. **Controle**: que é executado nos celulares e possui uma instancia do uOS que o permite se comunicar diretamente com o *Run Fast!*.
3. **Dois minijogos**: o Bônus e o *Break* que são pequenos jogos executados nos celulares durante a corrida. Por sua vez, estes foram incorporados a aplicação do controle.

Os minijogos e controle foram desenvolvidas na linguagem Android [10] e o jogo principal na linguagem Java [13], todos utilizando o paradigma de Orientação a Objetos. Estas linguagens foram escolhidas pela compatibilidade com o *middleware* uOS (Seção 2.1.2) e pela sua grande variedade de plataformas suportadas. Para o desenvolvimento do jogo principal, foram utilizadas as bibliotecas de GUI (*Graphic User Interface*) básicas do Java: AWT(*Abstract Window Tool-kit*) e *Swing*. Foram utilizadas estas bibliotecas por serem de fácil utilização, por permitirem a renderização de imagens bidimensionais, multiplataforma e por serem padrão do Java.

Para integrar os dispositivos e possibilitar a comunicação entre eles foi utilizado o *middleware* uOS juntamente com o conjunto de protocolos uP. Em cada dispositivo deve ter uma instância do *middleware* executando juntamente com os *drivers* desenvolvidos para estabelecer a comunicação. Além disso, foi utilizado o *plugin* de *Sockets* (TCP - *Transmission Control Protocol*), que utiliza uma conexão *wi-fi* para que os dispositivos

se comuniquem, visando obter uma velocidade de transmissão maior entre os dispositivos e manter a fluidez do jogo.

## 5.1 *Engine* do *RunFast!*

A *engine* foi desenvolvida de forma a fornecer certas funcionalidades básicas do jogo. As principais funcionalidades são o *game loop*, que faz o controle central dos estados do jogo, e o controle de *inputs*, que permite abstrair os *inputs*. Dentro do *game loop* estão as classes que podem ser estendidas pelo código específico do jogo (***GameObject***, ***Animation***), classes capazes de abstrair a comunicação gráfica (***Window***) e classes de controle do fluxo do jogo (***State***, ***StateManager***). Já no controle de *inputs* estão as classes para receber e tratar os *inputs* do jogo (***InputManager***, ***InputEvent***, ***InputListener***). As seções a seguir detalham estes componentes da *engine*.

### 5.1.1 *Game Loop* e Fluxo de Jogo

Um jogo é uma aplicação que está constantemente mudando de acordo com os vários eventos que ocorrem durante sua execução. Para isto, cria-se uma sequência de instruções que são executadas repetidamente, gerando um *loop* que é chamado de *game loop*. Desta mesma forma, o jogo *Run Fast!* também apresenta um *game loop*.

A classe *StateManager* controla o *game loop*. Antes de executar o *loop* principal cria-se o estado inicial e em seguida a janela do jogo através da classe *Window*. Isto faz com que seja apresentada a tela de abertura do jogo, no entanto ainda sem que seja possível qualquer tipo de interação. Então inicia-se o *game loop*. Assim como num filme, a sensação de movimento de um jogo acontece devido a apresentação de várias imagens sucessivas, chamados de quadros ou *frames*, num pequeno intervalo de tempo. Cada laço do *game loop* gera um quadro e o desenha na tela. Sendo assim, o *game loop* é composto pelos seguintes passos:

1. **Início do tempo do *frame***: é armazenado, em milisegundos, o momento do clock do processador em que este frame começou;
2. **Procedimentos de atualização do jogo (*update*)**: as movimentações do jogo são processadas e os valores dos elementos são atualizados. Os processos são feitos baseados no tempo do *frame* anterior, desta forma mantém-se uma fluidez dos acontecimentos do jogo de acordo com o tempo real percorrido. Nesta etapa, pode ocorrer uma mudança de estado (*State*), caso a corrida termine, por exemplo. Neste caso, há uma troca do modo de jogo para o estado de fim onde é mostrado o ranking dos vencedores. Para isto, instancia-se o novo estado e solicita-se a troca de estados a *Window*.
3. **Processos de renderização**: onde os elementos do jogo são redesenhados na tela de acordo com os novos valores. Nesta etapa que ocorre efetivamente a geração de um novo quadro.
4. **Cálculo da duração do *frame***: recupera o tempo final do *frame* e decrementa esta valor do tempo obtido no seu início. Caso o *frame* demore menos do que

o esperado deixa-se em modo de espera até alcançar o tempo de fps (*frames* por segundo) padrão, equalizando o tempo do *frame*.

Quando receber o comando de fim de jogo o *game loop* para de executar e a *StateManager* finaliza a aplicação.

A classe *State* apresenta 4 métodos abstratos que a *StateManager* utiliza para controlar o fluxo do jogo:

1. *load(DevicesController devController)*: é o método de carregamento onde o estado carrega os recursos necessários, como imagens. Este método é executado antes de se efetivar a troca de estados;
2. *unload()*: utilizado para descarregar alguns recursos que precisem de ser terminados antes que a instância seja marcada para coleta pelo Coletor de Lixo do Java (*Garbage Colector*). Este método é chamado imediatamente antes de ser efetivada uma troca de estados;
3. *update(int dt)*: responsável por realizar os *updates* das entidades do estado;
4. *render()*: o método que cria a imagem deste quadro, desenhando na tela seus elementos.

Além do *game loop* a classe *StateManager* controla também o fluxo de jogo. A estrutura criada para controlar este fluxo está representada na Figura 5.1. Esta arquitetura cria também uma abstração das características básicas da apresentação gráfica do jogo. Para realizar esta tarefa foi criada a classe *Window* e a classe *State* que estendem respectivamente as APIs *JFrame* e *JPanel* fornecidas pelo Java.

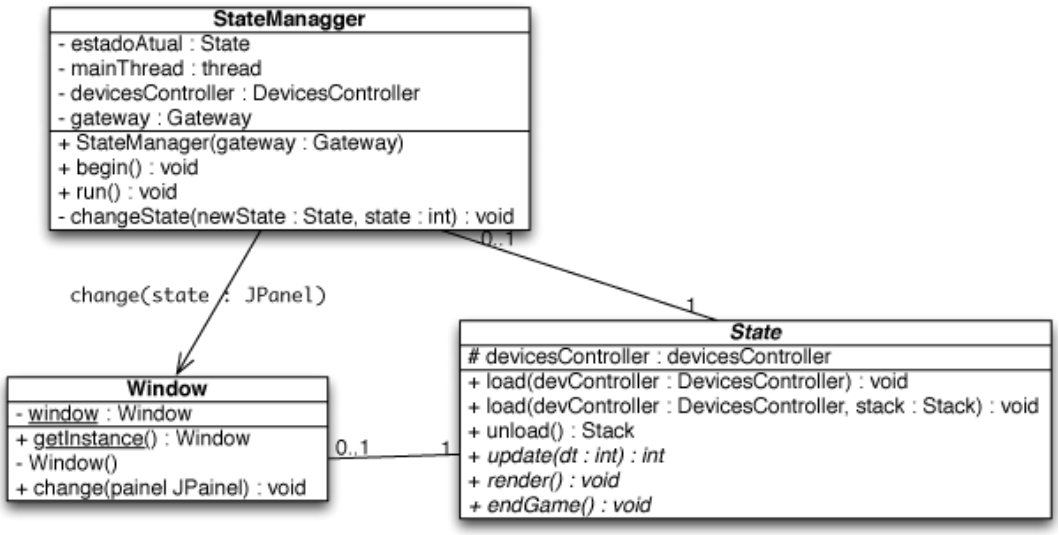



Figura 5.1: Diagrama de classes do controle de fluxo do jogo.

A classe *Window* cria uma janela para conter o jogo. Já a classe *State* estende o *JPanel*, que é o ambiente onde serão desenhados os elementos do jogo. Por fim, a *StateManager* faz o controle do fluxo. Para isto, primeiramente ela inicia a classe *Window* e uma classe filha da classe *State* que represente o estado inicial. Assim cria uma janela e o conteúdo inicial do jogo. Depois, durante o *loop* principal devem ocorrer trocas de estados do jogo. Nestas trocas, os novos estados são passados para a classe *Window*, pela *StateManager*, trocando o *JPanel* mostrado na janela do jogo.

### 5.1.2 Controle de *Inputs*

Os componentes envolvidos para se capturar as entradas dos usuários são representados na Figura 5.2. O objetivo desta estrutura é padronizar os *inputs* do jogo. Esta arquitetura utiliza o padrão de projeto de software *Observer* [9], que cria uma dependência de um objeto para muitos. Neste padrão de projeto, os objetos observadores (*Listeners*) se inscrevem no objeto a ser observado (*Observer*) que, por sua vez, os notifica quando esses eventos ocorrerem.

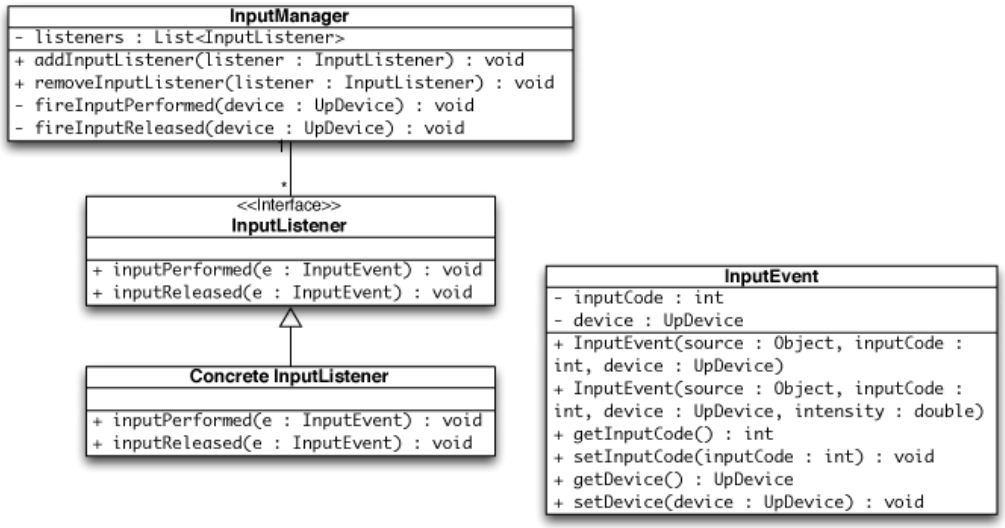



Figura 5.2: Diagrama de classes de controle de *inputs*.

Os eventos são representados através da classe *InputEvent*. Ao ocorrer um comando gerado pelo usuário um objeto é instanciado e são passadas as informações do tipo de *input* e o dispositivo que o realizou. Os *inputs* podem ser os seguintes:

1. Entradas de movimentação, que identifiquem a direção de um movimento, como comandos de direção para cima ou para baixo;
2. Entradas cujo objetivo, ao serem disparadas, seja realizar uma ação, como atirar ou deixar uma bomba;

3. Por fim, entradas para execução de ações extras, como acionar um item e determinar sobre qual time seus efeitos devem ser atribuídos.

A *InputManager* é a classe que recebe os eventos de *input* dos dispositivos. Esta classe cria um *InputEvent* e notifica seus *listeners*, utilizando a interface *InputListener* que possui os métodos que serão acionados para notificar os *inputs*.

## 5.2 Componentes do *RunFast!*

Vários elementos do jogo podem ser observados na Figura 5.3, como itens, carros, informações dos times, mapa do jogo, etc. A construção destes componentes é permitida pela utilização da *engine* desenvolvida. Dessa forma, tanto a mecânica do jogo (estados, fluxo de execução, divisão de times) quanto itens e equipamentos são construídos utilizando os componentes da *engine*. A seguir são apresentados detalhes de implementação destes componentes.

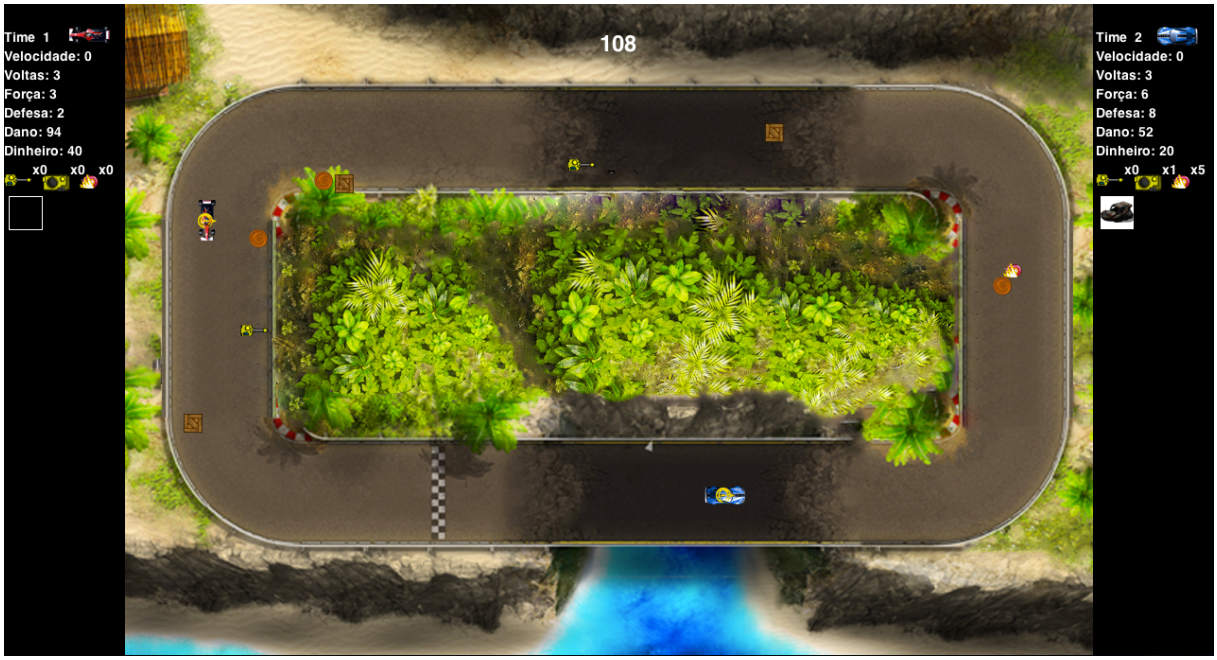



Figura 5.3: O jogo *Run Fast!* durante uma corrida.

### 5.2.1 Estados do Jogo

O jogo é composto por 4 estados (Figura 5.4):

- O *StateMenu* responsável por apresentar ao jogador as opções de jogo disponíveis. Na implementação atual são apresentados as opções de jogar (*Play*) e sair do jogo (*Quit*). Este também é o estado inicial do jogo;

- O *StateSelection* é onde os pilotos podem escolher o carro do time. Quando todos os pilotos escolherem um carro o jogo é iniciado. Este estado ocorre logo depois de selecionar a opção *"Play"* no *StateMenu*;
- O *StateGame*, estado em que ocorre a corrida, onde os jogadores controlam os carros em busca de fazer do seus times o primeiro colocado. Este estado ocorre assim que todos os times escolhem um carro ou quando no *StateWin* escolhem a opção *"Restart"*.
- O *StateWin* é onde se mostra o *ranking* final dos times e um menu com as opções de reiniciar a corrida (*Restart*), voltar para o menu (*Menu*) e a opção de sair do jogo (*Quit*). Este estado ocorre assim que a corrida termina (quando o tempo acaba).

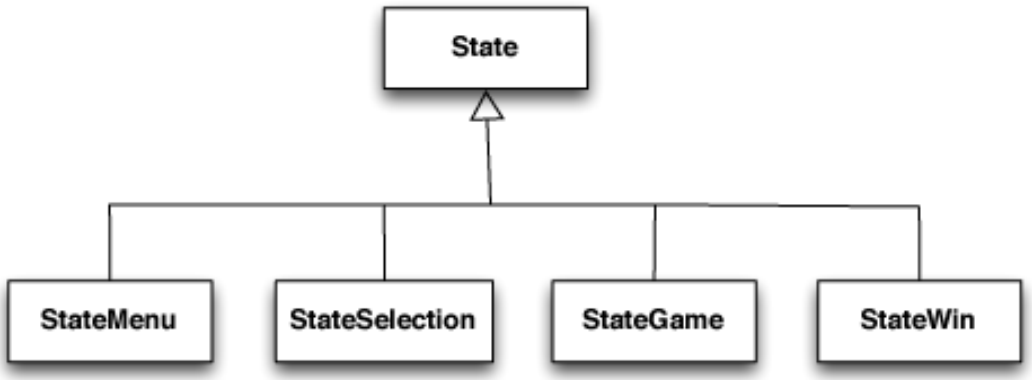



Figura 5.4: Diagrama de classes dos estados do jogo.

### 5.2.2 Times e Carros

Os elementos principais do jogo são os carros que são controlados em time (piloto, copiloto e assistentes). Por este motivo, foram construídas duas classes: a classe *Car* e a *Team*. Estas classes estão representadas na Figura 5.5. A classe *Car*, diferentemente da *Team* que é uma classe de controle, estende a *GameObject* da *engine*. A *GameObject* é uma classe abstrata da *engine* criada para as entidades do jogo (carros, itens, equipamentos, etc). Possuindo os dados de posição do objeto na tela e oferecendo metodos abstratos de atualização e renderização.

A classe *Team* agrupa os jogadores de uma mesma equipe e separa as ações de cada um. Assim que um jogador entra no jogo ele é adicionado a um time existente, em um personagem vago, ou ele cria um novo time, tornando-se o piloto. A classe *Team* diferencia cada personagem por dispositivo de entrada. Além disso, esta classe estende a interface *InputListener* da *engine*. Assim, a partir das informações de qual dispositivo está vinculado cada personagem é feita uma separação dos eventos recebidos e repassados ao carro. Desta forma, pode-se utilizar controles semelhantes para cada personagem e ainda assim garantir que as funções de um personagem não sejam executadas por outro.

Os *inputs*, recebidos pelos dispositivos do jogo, são devidamente filtrados pela *Team* e redirecionados para a instancia de *Car* em forma de ações. Assim, a *Car* se responsabiliza

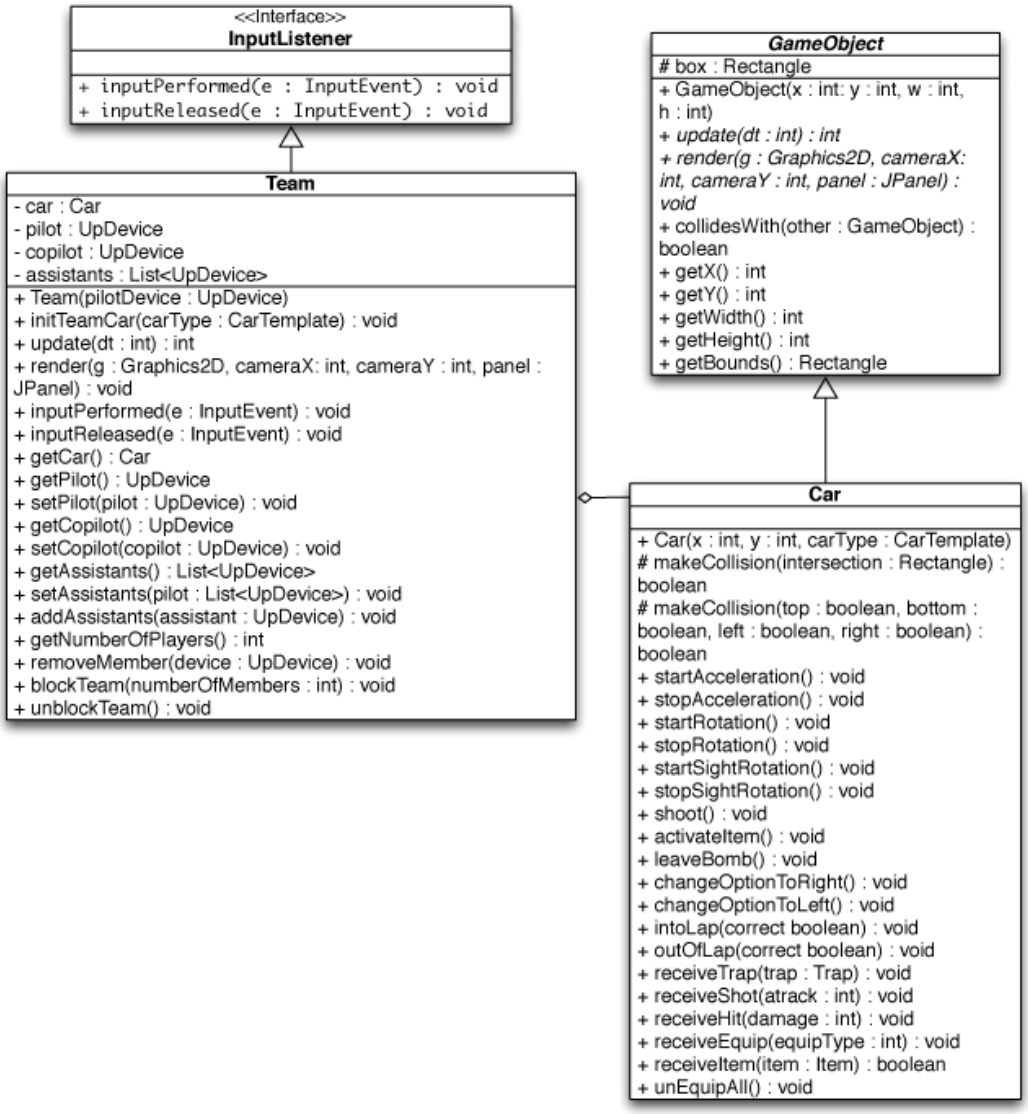



Figura 5.5: Diagrama de classes da interação entre as classes *Team* e *Car*.

apenas em fornecer as funcionalidades do carro, como *startAcceleration()* (ação do piloto), *leaveBomb()* (ação do copiloto), etc.

### 5.2.3 Equipamentos, Itens e Armadilhas

Os equipamentos, os itens e as armadilhas são implementados, respectivamente, pelas classes *Equip*, *Item* e *Trap*. Assim como a *Car* estas classes estendem a classe *GameObject* e são tratadas como objetos que são renderizados numa determinada posição da tela. Na Figura 5.6 ilustra-se a presença destas entidades no jogo.

### 5.2.4 Mapa

O mapa tem acesso a todas as entidades do jogo. O mapa é criado pela classe *Map* e tem como objetivo organizar e verificar as relações entre os diversos elementos do jogo,

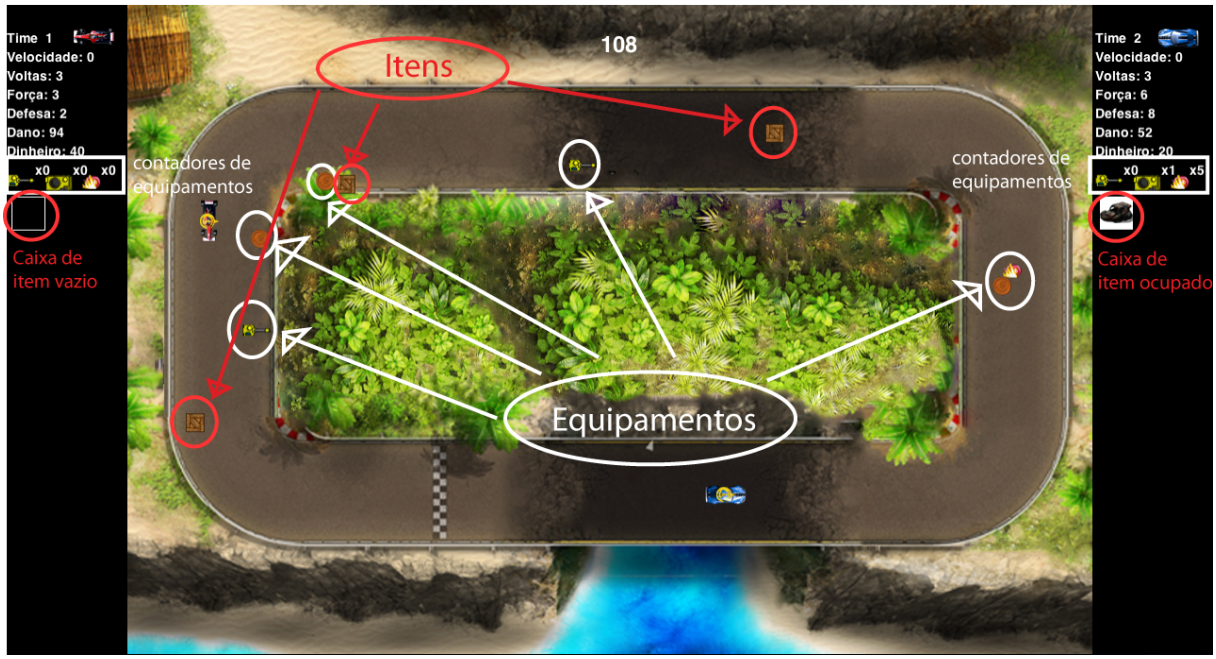



Figura 5.6: Ilustração dos equipamentos e dos itens durante o jogo.

sendo responsável por executar as regras do mundo do jogo. As principais responsabilidades do *Map* são:

- **Gerar os itens e equipamentos**: periodicamente o mapa gera aleatóriamente itens e equipamentos distribuídos por sua extensão de acordo com o número de times no jogo.
- **Cálcular colisões**: o mapa verifica constantemente as colisões entre as entidades (itens, equipamentos, armadilhas, etc) com os carros. Além de verificar também as colisões entre os carros e entre estes e os limites da pista.
- **Verificar o fim de jogo**: o mapa está sempre verificando se o tempo limite do jogo foi alcançado para saber se este deve ser finalizado.

## 5.3 Integração do uOS

Para acessar os dispositivos móveis no ambiente o jogo utiliza o *middleware* uOS. Este *middleware* utiliza uma rede, no caso do *Run Fast!* o *wi-fi*, para transmitir mensagens entre os dispositivos. Para mandar estas mensagens, utiliza-se os *drivers* e seus serviços. Quando o *middleware* é iniciado num dispositivo, este inicia os *drivers* e as aplicações (*Applications*) que serão executadas sobre o *middleware* e disponibiliza os *drivers* e seus serviços na rede. Assim, quando um novo dispositivo que tenha o *middleware* executando entrar no ambiente, é realizado o chamado *handshake*, que é quando dois dispositivos trocam suas informações e de seus *drivers*. Uma vez realizado o *handshake* entre dois dispositivos, suas aplicações podem usufruir dos serviços oferecidos pelos *drivers* um do

outro. Sendo assim, o *middleware* uOS é utilizado como base para o desenvolvimento de tais aplicações (Figura 5.7).

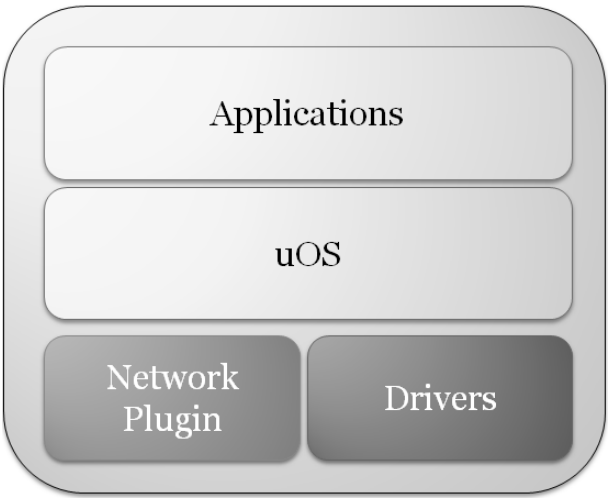



Figura 5.7: Visão geral do funcionamento do *middleware* uOS.

No *Run Fast!* o jogo principal que executa no computador inicia o *middleware* assim que é iniciado, já nos aplicativos dos celulares existe um botão de ligar e desligar o *middleware*. Uma vez que ambos os *middlewares* estejam executando, as aplicações do *Run Fast!* utilizam os *drivers* *RFDevicesDriver* e o *RFControllerDriver* para se comunicarem.

Na Figura 5.8 mostra-se a organização geral da arquitetura do jogo para estabelecer as comunicações entre os dispositivos. Esta arquitetura pode ser subdividida em duas partes principais: a transmissão de *inputs* realizados pelo jogador e o controle de entrada e saída de dispositivos do jogo. A arquitetura mantém um *driver* em cada lado da comunicação dos dispositivos. Possibilitando uma comunicação de duas vias e deixando mais fácil das partes se encontrarem, uma vez que cada um possui um único *driver* específico. Estas estruturas serão apresentadas nas seções que se seguem.

### 5.3.1 Transmissão de *Inputs* do Controle

A Figura 5.9 mostra o diagrama das classes que estabelecem a comunicação entre os dispositivos quanto a transmissão de eventos de *input* do controle ao jogo.

O *RFControllerDriver* estende a classe *UosEventDriver*. Desta forma, este *driver* possibilita que outras classes se registrem para receber notificações. A classe *ControllerListener* observa os eventos de *input* criados pela classe *ControllerActivity* e chamam os serviços de *inputPerformed* e *inputReleased* oferecidos pelo *RFControllerDriver*. Assim, o *RFControllerDriver* notifica todos os seus ouvintes da ocorrência deste *input*.

A *InputManager* implementa a classe *UosEventListener*, podendo ouvir eventos gerados por classes *UosEventDriver*. Quando um dispositivo entra no jogo a *InputManager* se registra no *RFControllerDriver* oferecido por este. Assim, recebe as notificações criadas por este *driver* e os repassa para seus *listeners* que vão tratá-los de acordo com o dispositivo que o gerou.

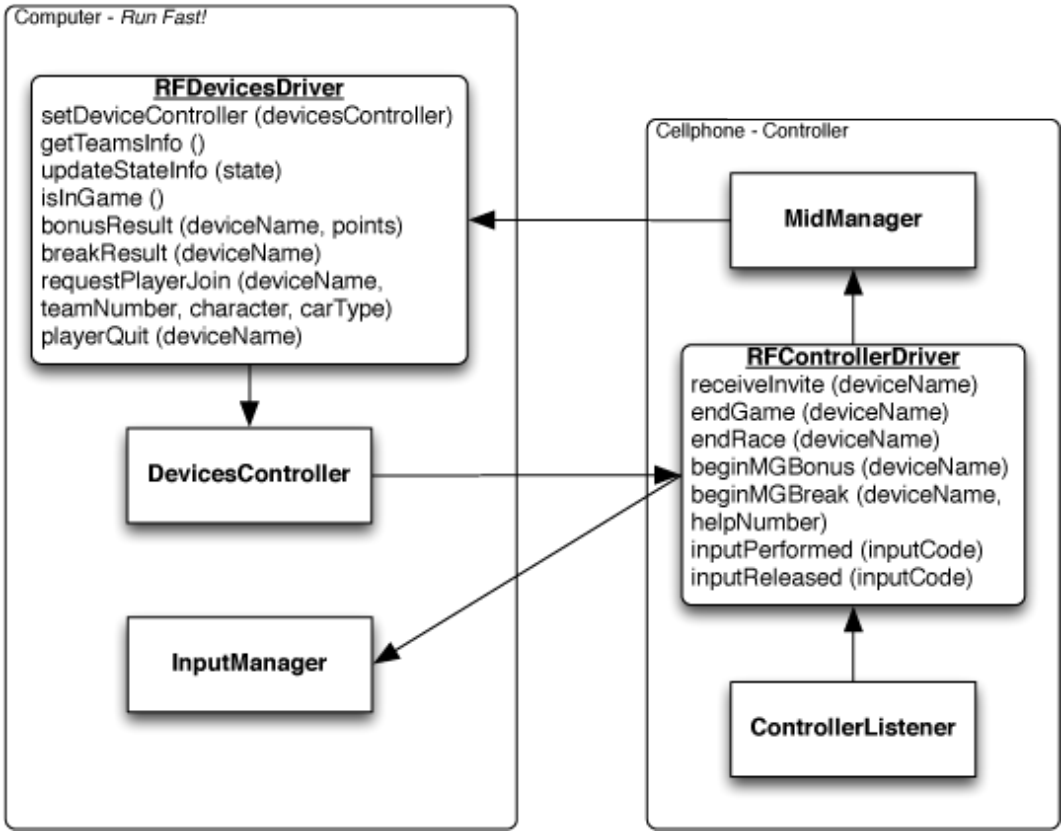



Figura 5.8: Organização geral das interações do *Run Fast!* utilizando o *middleware* uOS.

### 5.3.2 Entrada e Saída de Dispositivos

Para controlar a entrada e saída de dispositivos são utilizados tanto o *RFControllerDriver* quanto o *RFDevicesDriver*. Estes *drivers* fornecem serviços de comunicação para que os dispositivos possam se comunicar e confirmar a entrada e saída de um novo celular no jogo. A Figura 5.10 ilustra a arquitetura de controle de entrada e saída de dispositivos dentro do jogo *Run Fast!*.

O *Run Fast!* utiliza a classe *DevicesController* para analisar os dispositivos que surgirem e deixarem o ambiente. Para isto, é feito uma constante averiguação da lista de dispositivos com o *RFControllerDriver*. Caso surja um dispositivo novo este recebe um convite do jogo. De toda forma, a aplicação do celular também permite que seja procurado um novo jogo. Ao sair do jogo, o celular informa o *Run Fast!* através do *RFDevicesDriver* sobre sua saída, disparando o processo de saída do dispositivo. Se por algum motivo o celular deixar o ambiente sem avisar sua saída do jogo, assim que sua retirada for percebida o *DevicesController* inicia o procedimento de saída.

Além da entrada e saída em si, esses *drivers* fornecem serviços auxiliares e serviços para iniciar e finalizar os minijogos. Os serviços auxiliares informam os estados do jogo principal e a composicão dos times atuais, de forma que o dispositivo que deseja entrar possa informar ao jogador qual time e personagem ele pode escolher. Já os serviços dos minijogos solicitam o início, ou informam o término de um determinado minijogo.

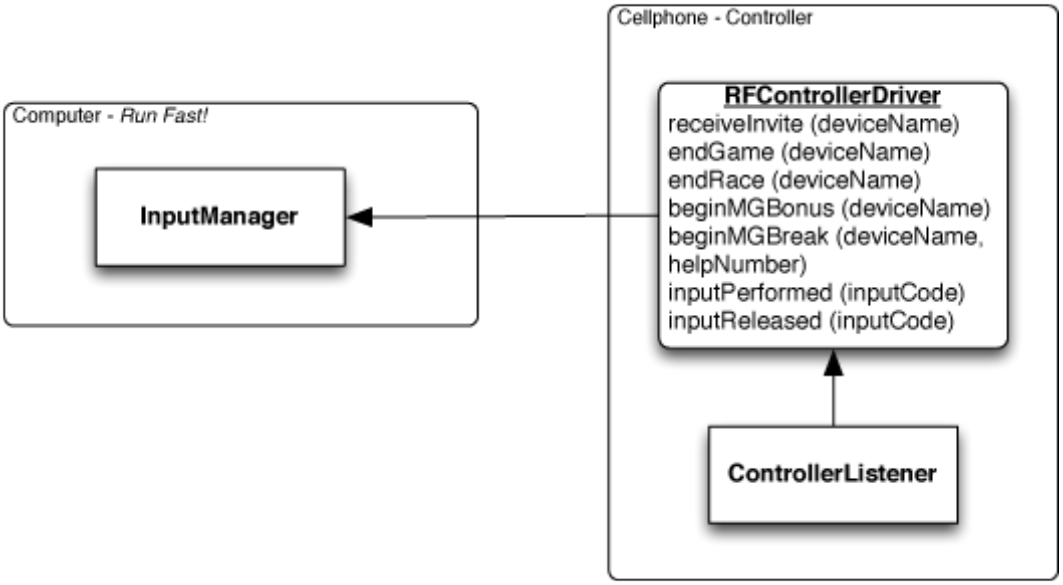



Figura 5.9: Organização das interações do *Run Fast!* relativa aos envios de *inputs*.

## 5.4 Controle

O controle é o aplicativo Android responsável por prover uma interface para a entrada do jogador ao jogo e, uma vez que esteja conectado, passar os comandos gerados pelo jogador ao *Run Fast!*. Para realizar esta tarefa, o controle apresenta 4 estados diferentes, que no Android são apresentados como classes que estendem a *Activity*, que é a classe do Android que cuida de criar uma janela no celular onde serão colocados os elementos da UI (*User Interface*):

1. *MainActivity*: este estado tem como objetivo fazer com que os dispositivos sejam encontrados. Ele permite ao usuário ligar e desligar o *middleware* (que deve estar previamente instalado no dispositivo), além de apresentar a interface de procurar e receber convites para um jogo no ambiente. Esta é a *Activity* inicial do controle.

2. *SelectActivity*: esta *Activity* é utilizada para que o jogador decida um personagem e um time para entrar no jogo. Para isto, esta *activity* se comunica com o jogo por duas vezes, uma ao ser criado, para receber as informações dos times, e outra quando o jogador fizer sua decisão, para solicitar ao jogo sua entrada. Esta *Activity* ocorre assim que o jogador decide entrar em um jogo.

3. *SelectCarActivity*: este estado é utilizado para que um piloto que criou um time durante a corrida possa escolher um carro. Ocorre, apenas, caso o jogador esteja criando um novo time durante o *StateGame*.

4. *ControllerActivity*: esta é a *Activity* que fornece a interface do controle ao usuário de acordo com o personagem escolhido. Além de passar os eventos de *input* gerados pelo jogador ao jogo, por meio do *driver RFControllerDriver*.

Para guardar os dados do *middleware* e controlar as ações dos estados, foi criado a classe *MidManager*. Esta classe tem uma instância única dentro da aplicação que é

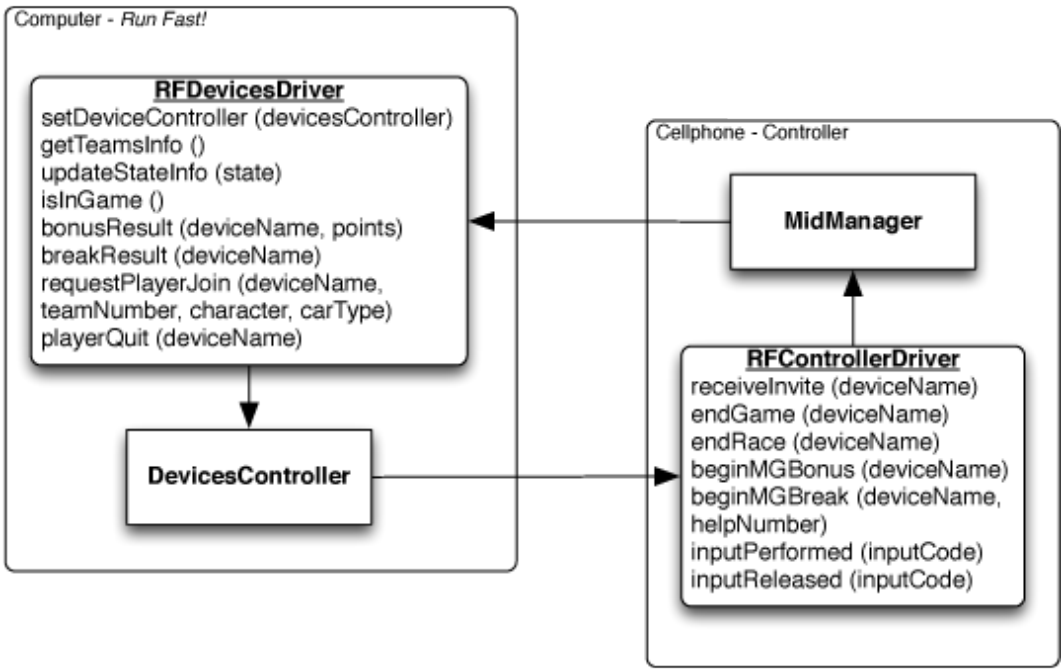



Figura 5.10: Organização das interações do *Run Fast!* relativo as entradas e saídas dos dispositivos.

utilizada por todos os estados do controle. Além disso, esta sempre é informada quando há uma mudança de *Activity*. Assim pode decidir que ação tomar quando for solicitada numa ação externa pelo jogo.

## 5.5 Minijogos

Os minijogos são desenvolvidos sobre a arquitetura do Android que já provê um controle de fluxo de estados. Apesar de básico, este fluxo é adequado dada a baixa complexidade dos minijogos. Estes minijogos são desencadeados por uma solicitação externa do jogo principal e ao terminarem fazem uma chamada de um serviço do *RFDevicesDriver* do jogo para realizar os procedimentos de término dos mesmos.

A *engine* destes jogos se resume a um *game loop* diretamente relacionado ao código específico, uma classe que disponibiliza os sons (*SoundEffect*) e a classe de animação (*AnimatedObject*), utilizado para algumas animações gráficas, além da classe *LoadManager* utilizada para carregar as imagens. Em ambos os jogos cria-se uma *Activity* (*BonusActivity* e *BreakActivity*, dependendo do jogo) que cria uma *GameSurface* onde se implementa o *game loop* e o código específico. A *GameSurface* implementa as regras do jogo e cria uma lista de *AnimetedObjects* de acordo com as regras de cada um dos minijogos. Esta estrutura está demonstrada na Figura 5.11.

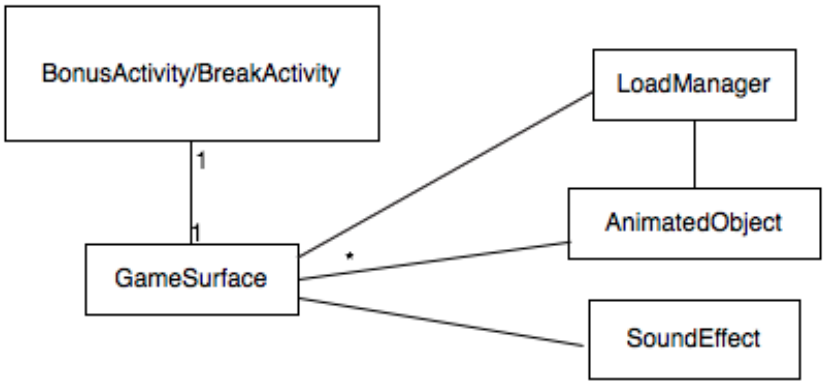



Figura 5.11: Diagrama de classes dos minijogos (Bônus e *Break*).

## 5.6 Experimentações

Foram realizados alguns testes de execução com o jogo *Run Fast!*. Mais precisamente, foram observados de forma qualitativa a percepção de entrada e saída de jogadores, a fluência da transmissão de *inputs* dos celulares para o computador e as capacidades comunicativas oferecidas pelos serviços dos *drivers*.

Para realizar estes testes foram utilizados um MacBook Pro com sistema operacional Mac OS X (Versão 10.6.8), processador 2.3 GHz Intel Core i5 e memória de 4 GB 1333 MHz DDR3, três celulares android: um HTC Nexus One com Android versão 2.3.6; um Motorola XT303 com Android versão 2.3.6; e um Motorola XT890 com Android versão 4.0.4; e dois dispositivos simulados pelo jogo (no próprio MacBook) cujas entradas acontecem via teclado. Além disso, foi utilizada uma conexão *ethernet*.

Para realização dos testes utilizou-se os dois dispositivos simulados como pilotos de duas equipes, enquanto os três dispositivos reais entraram de forma arbitrária no jogo durante diferentes momentos, avaliando a comunicação entre o jogo e os celulares. Após isso os jogadores realizaram suas ações dentro do jogo, onde verificou-se a usabilidade dos *inputs* gerados pelos celulares. Por fim, foram realizados alguns testes de entrada e saída de dispositivos no ambiente, avaliando o reconhecimento dos mesmos.

A qualidade da comunicação entre os dispositivos móveis e o computador, tal como o tempo de resposta obtido dos *inputs* dos jogadores utilizando os três dispositivos móveis praticamente não apresentaram atraso entre o comando do usuário e a resposta do jogo ao comando. A jogabilidade oferecida pelos *inputs* do controle foram até mesmo próximas a jogabilidade dos *inputs* aplicados no teclado. Apresentando algumas poucas dificuldades apenas quanto a utilização do *touch* do celular para realizar tais ações, onde em algumas vezes os botões do controle travassem precisando de um segundo clique para voltar ao normal. Além disso, há uma pequena demora, de aproximadamente dois segundos, ao serem disparados os minijogos, o que é explicado pelo fato de ser necessário que o celular crie uma nova *activity* carregando os elementos do minijogo.

Quanto a entrada e saída de jogadores, uma vez que os dispositivos se encontraram no ambiente a troca de mensagens ocorre sem problemas, facilitando a entrada e saída dos jogadores e os acontecimentos durante o jogo. No entanto, a ocorrência da entrada ou

saída de um novo dispositivo no ambiente requer um tempo de espera de aproximadamente dois minutos até que o jogo perceba efetivamente a mudança, devido ao tempo de espera de rastreamento de novos dispositivos pelo radar do *middleware*, o que é um tempo de espera alto.

# Capítulo 6

# Conclusão

Encorajado pela distribuição de dispositivos, principalmente o do uso de celulares, foi desenvolvido o jogo *Run Fast!*. Este jogo serve de caso de estudo do *middleware* uOS e foi criado visando usar suas funcionalidades. O *Run Fast!* tem como objetivo abrir caminho para a criação de outras aplicações úbiquas com o uOS, uma vez que ambientes úbiquos fornecem diversas possibilidades de desenvolvimento de novas aplicações. O próprio funcionamento do jogo proporcionou a avaliação de vários quesitos do *middleware*, que foram sendo aperfeiçoados, com a equipe responsável. Consequentemente, o *Run Fast!* serve como validação da utilização do *middleware* na criação de aplicações úbiquas. Uma vez que, por meio deste, verifica-se a capacidade de reconfiguração dinâmica de acordo com os dispositivos presentes no ambiente e a eficiência da comunicação entre eles, de forma cooperativa, possibilitando que sejam utilizados até mesmo como dispositivos de entrada.

Como sugestões de trabalhos futuros, seguem algumas melhorias e expansões no jogo desenvolvido e no uOS:

- Neste projeto, foi utilizado o *EthernetPingRadar* para descoberta da entrada e saída de dispositivos no ambiente. No entanto, é necessário um tempo significativo para reconhecer uma mudança deste tipo no ambiente. Abrindo espaço para uma melhora no sistema do uOS.
- O *middleware* apresenta, ainda, uma baixa quantidade de documentação, tornando difícil o aprendizado. Como trabalhos futuros, pode-se desenvolver uma documentação com o intuito de ensinar a utilizá-lo.
- Baseado na aplicação do *Run Fast!* para o Android, pode-se desenvolver uma interface gráfica do *middleware* que possibilite ao usuário instalar os aplicativos que devem executar sobre ele.
- No jogo, tanto os minijogos, quanto o controle, estão instalados como uma única aplicação associada a instância do *middleware* no celular. Para futuros trabalhos, pode-se procurar utilizar mobilidade de código para transmitir essas aplicações em tempo real, ou até mesmo para que alguém novo no ambiente possa baixa-lo.
- O jogo *Run Fast!* em sim pode ser melhorado, fazendo um balanceamento do jogo, ou criando, ainda, a possibilidade de jogar contra o computador, desenvolvendo uma IA (Inteligência Artificial).

- Usando o funcionamento do *Run Fast!* como caso de estudo, pode-se desenvolver uma *Game Engine* para jogos úbiquos utilizando o uOS.
- O *Run Fast!* pode ser evoluído quanto as formas de *input*, onde pode-se variar os controles de acordo com modelos de celulares, ou pode-se utilizar outros dispositivos, como tablets, dispositivos de reconhecimento gestuais.
- Podem ser criados outros tipos de minijogos, que utilizem mais recursos fornecidos pelos celulares, ou minijogos mais interativos que se comuniquem entre si.

# Referências


[1] Steve Benford, Andy Crabtree, Martin Flintham, Adam Drozd, Rob Anastasi, Mark Paxton, Nick Tandavanitj, Matt Adams, and Ju Row-Farr. Can you see me now? *ACM Trans. Comput.-Hum. Interact.*, 13(1):100–133, March 2006. 2

[2] Staffan Björk, Jennica Falk, Rebecca Hansson, and Peter Ljungstrand. Pirates! using the physical world as a game board. In *In Proceedings of Interact 2001*, pages 9–13, 2001. 2

[3] Staffan Björk, Jussi Holopainen, Peter Ljungstrand, and Karl-Petter AAkesson. Designing ubiquitous computing games – a report from a workshop exploring ubiquitous computing entertainment. *Personal Ubiquitous Comput.*, 6(5-6):443–458, January 2002. 1

[4] F. Buzeto, A. Helena Castillo, C. Castanho, and J. Ricardo. What is going on with ubicomp games. In *Brazilian Symposium on Computer Games and Digital Entertainment - SBGAMES*, November 2012. 2, 8

[5] Fabricio Nogueira Buzeto. Um conjunto de soluções para a construção de aplicativos de computação ubíqua. Mestrado, Universidade de Brasília, UNB, Julho 2010. 3, 4, 5

[6] Fabricio Nogueira Buzeto, Carlos Botelho de Paula Filho, Carla Denise Castanho, and Ricardo Pezzuol Jacobi. Dsoa: a service oriented architecture for ubiquitous applications. In *Proceedings of the 5th international conference on Advances in Grid and Pervasive Computing*, GPC'10, pages 183–192, Berlin, Heidelberg, 2010. Springer-Verlag. vii, 1, 5, 6

[7] Adam Drozd, Steve Benford, Nick Tandavanitj, Michael Wright, and Alan Chamberlain. Hitchers: designing for cellular positioning. In *Proceedings of the 8th international conference on Ubiquitous Computing*, UbiComp'06, pages 279–296, Berlin, Heidelberg, 2006. Springer-Verlag. vii, 2, 12, 13

[8] Martin Flintham, Rob Anastasi, Steven Benford, Adam Drozd, James Mathrick, Duncan Rowland, Amanda Oldroyd, Jon Sutton, Nick Tandavanitj, Matt Adams, and Ju Row-Farr. Uncle roy all around you: mixing games and theatre on the city streets. In Copier Marinka and Raessens Joost, editors, *Level Up Conference Proceedings: Proceedings of the 2003 Digital Games Research Association Conference*, Utrecht, November 2003. University of Utrecht. vii, 2, 10, 11

[9] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995. 33

[10] Google. Android. http://www.android.com/. 30

[11] Jane Evelyn Mcgonigal. *This might be a game: ubiquitous play and performance at the turn of the twenty-first century*. PhD thesis, Berkeley, CA, USA, 2006. AAI3253985. 7

[12] Eduardo Moreira. Apple lista os aplicativos mais baixados de todos os tempos na app store. http://www.techtudo.com.br/noticias/noticia/2012/03/apple-lista-os-aplicativos-mais-baixados-de-todos-os-tempos-na-app-store.html, Março 2012. 1

[13] Oracle. Java. http://www.java.com/pt_BR/. 30

[14] Daniel Pargman and Peter Jakobsson. Five perspectives on computer game history. *interactions*, 14(6):26–29, November 2007. 1, 7

[15] Google Play. Top paid in apps. https://play.google.com/store/apps/collection/topselling_paid. 1

[16] Kay Römer and Svetlana Domnitcheva. Smart playing cards: A ubiquitous computing game. *Personal Ubiquitous Comput.*, 6(5-6):371–377, January 2002. vii, 2, 16, 17

[17] Bruce Thomas, Ben Close, John Donoghue, John Squires, Phillip De Bondi, and Wayne Piekarski. First person indoor/outdoor augmented reality application: Arquake. *Personal Ubiquitous Comput.*, 6(1):75–86, January 2002. vii, 2, 14, 15

[18] M. Weiser. The computer for the 21st century. *IEEE Pervasive Computing*, 1(1):19–25, January 2002. 1, 3

[19] Mark Weiser. The world is not a desktop. *Interactions*, 1(1):7–8, January 1994. 4

[20] Mark Weiser and John Seely Brown. Designing calm technology. December 1995. 4